

M 1 101 – Systèmes d'exploitation – Architecture des microprocesseurs

Année Spéciale, année 2019 – 2020

Département Informatique

IUT de Bordeaux

Contenu du cours M 1 101

1. Histoire de l'informatique
2. Codage de l'information
3. Système d'exploitation
 1. Introduction aux systèmes d'exploitation
 2. Le cours
 3. Environnement de travail
 4. Utiliser Linux
 5. Lignes de commandes
 6. Arborescence
 7. Commandes utiles

Contenu du cours M 1 101 (suite)

3. Système d'exploitation (suite)

- 8. Compression
- 9. Archivage
- 10. Redirection
- 11. Droits d'accès
- 12. Commandes de filtrage

4. Shells scripts

- 1. Variables, paramètres, expressions
- 2. Fonctions
- 3. Arithmétique
- 4. Structure de contrôle « case »
- 5. Processus

Contenu du cours M 1 101 (suite)

- 6. Boucle for
- 7. Décisions
- 8. Boucle while

1. Histoire de l'Informatique

-**3000** : Période de l'empereur Chinois **Fou-Hi** dont le symbole magique, l'octogone à trigramme contient les 8 premiers nombres représentés sous forme binaire par des traits interrompus ou non : 000 001 010 011 etc...



-**500** : Moyen Orient : le boulier



Mécanisation du calcul Schickard (1623), Pascal (1642), Leibniz (1673) → réalisation des additions, soustractions, multiplications et mémorisation des résultats intermédiaires grâce à des systèmes mécaniques tels que des roues dentées



Automatisation du travail Falcon (1728), Jacquard (1805) métier à tisser utilisant des cartes perforées

Calcul automatique Babbage (1833) **Babbage** imagine et tente de réaliser une **machine à différences** puis une **machine analytique** qui contient les concepts de ce que sera l'ordinateur moderne : unité de calcul, mémoire, registre et entrée des données par carte perforée.



1. Histoire de l'Informatique

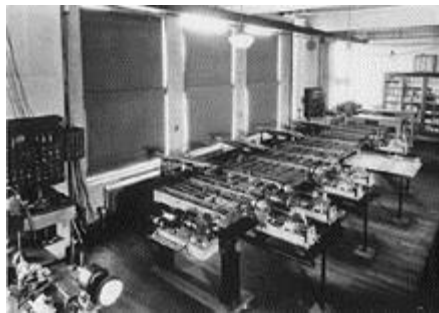
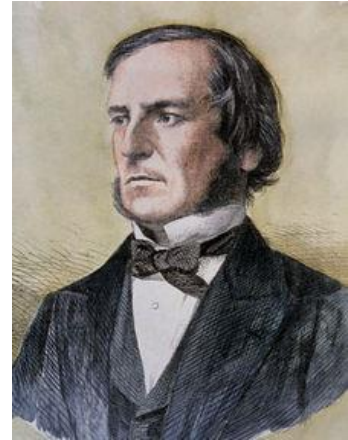
1854 : **Boole** publie un ouvrage dans lequel il démontre que tout processus logique peut être décomposé en une suite d'opérations logiques (ET, OU, NON) appliquées sur deux états (ZERO-UN, OUI-NON, VRAI-FAUX, OUVERT-FERME)

Hollerith (1884) crée une tabulatrice à cartes perforées

1904 : Invention du premier tube à vide, la **diode** par John Fleming

1924 : La firme créée par Hollerith en 1896, est renommée International Business Machine

1935 : **IBM** commercialise l'**IBM 601**, un calculateur à relais utilisant des cartes perforées capable de réaliser une multiplication en une seconde



1. Histoire de l'Informatique

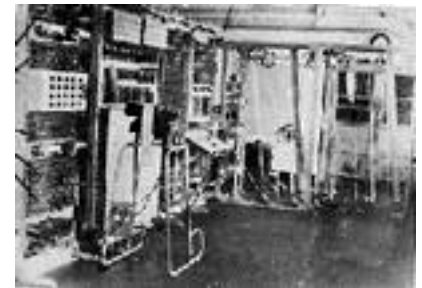
1937 : Alan Turing publie un document sur les nombres calculables (Machine de Turing)

1938 : Thèse de **Shannon** → le parallèle entre les circuits électriques et l'algèbre Booléenne. Il définit le chiffre binaire : **bit** (BInary digiT).

1938 : Création du premier ordinateur binaire programmable mais mécanique « **Versuchmodell 1** » ou **Z1** par **Konrad Zuse**

1939 : Réalisation d'un deuxième ordinateur, le **Z2** en remplaçant une partie des pièces mécaniques du **Z1** par des relais électromécaniques de téléphone (puis Z3 premier véritable ordinateur détruit en 1945 et Z4, Z1 et Z4 en photo)

1940 : les calculateurs **Robinson** et **Colossus** avec les concepts d'arithmétique binaire, d'horloge interne, de mémoire tampon, de lecteurs de bande, d'opérateurs booléens, de sous programmes et d'imprimantes.



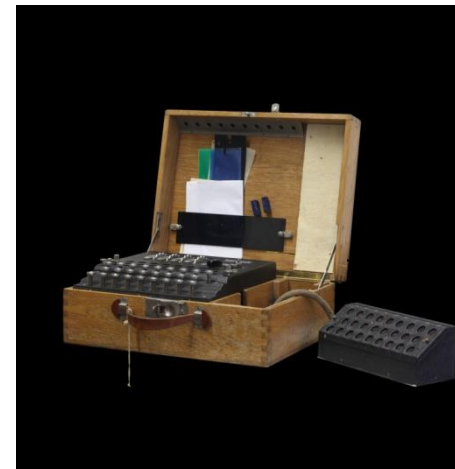
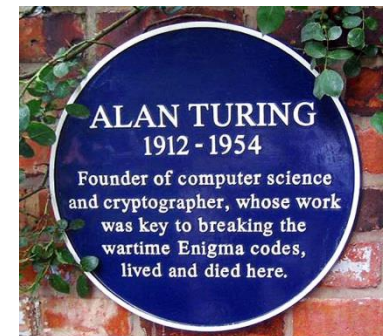
1. Histoire de l'Informatique

1940 : Pour décrypter les messages de l'armée allemande, les Anglais mettent au point sur le site de **Bletchey Park**, les premières machines qui intègrent les concepts d'arithmétique binaire, d'horloge interne, de mémoire tampon, de lecteurs de bande, de sous-programmes et d'imprimantes. « Secret défense » jusqu'en 1975.

1945 : **Alan Turing** a joué un rôle majeur dans la victoire des Alliés (gain de deux ans de guerre), mais marginalisé par la société et réhabilité par la reine Élisabeth II en 2013.

De nombreux films et documentaires relatent cet épisode historique:

Imitation Game (2014), La drôle de guerre d'Alan Turing (2014), U571 (2000), John Von Neuman, prophète du XXI ème siècle (2015), Enquêtes codées (Série britannique 2012-2013) ...



1. Histoire de l'Informatique

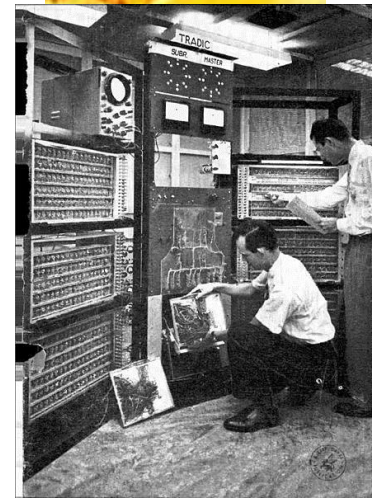
1945 : **John Von Neuman**, ayant rejoint l'équipe travaillant sur l'[ENIAC](#), publie le premier rapport décrivant ce que devrait être un ordinateur à programme enregistré → **architecture Von Neuman**.

1946 : Création de l'**ENIAC** (Electronic Numerical Integrator and Computer) par **P. Eckert** et **J. Mauchly**. Un calculateur composé de 19000 tubes pèse 30 tonnes, occupe une surface de 72 m² et consomme 140 kilowatts. Horloge : 100 KHz. Vitesse : environ 330 multiplications par seconde.

Décembre 1947 : Invention du **transistor** par **William Bradford Shockley**, **Walter H. Brattain** et **John Bardeen** dans les laboratoires de Bell Telephone.

1956 : Création du premier **ordinateur à transistors** par la Bell : le **TRADIC** qui amorce la seconde génération d'ordinateurs.

1957 : Création du premier langage de programmation universel, le **FORTRAN** (FORmula TRANslator) par **John Backus** d'**IBM**.



1. Histoire de l'Informatique

1959 : Digital crée le **PDP-1**, le premier mini ordinateur commercial interactif



Novembre 1971 : Intel commercialise le premier micro ordinateur **MCS-4** basé sur son tout nouveau microprocesseur 4004 et contenant aussi une Rom Intel 4001, une Ram Intel 4002 et un registre à décalage Intel 4003.



Années 90 accession des micro-ordinateurs au grand public, émergence du multimedia, d'internet, des jeux. Quasi monopole imposé et arrogant des PC (Personal Computer) et Microsoft.

Années 2000 introduction de l'ordinateur dans les activités quotidiennes de chacun, aide à la conduite automobile interactive, téléphonie portable, cuisine assistée, choix de programmes télévisés personnalisés...

DE NOS JOURS

Processeurs de plus en plus complexes : plusieurs coeurs, des lignes de caches, des coprocesseurs etc.

Évolution du nombre de transistors par processeur

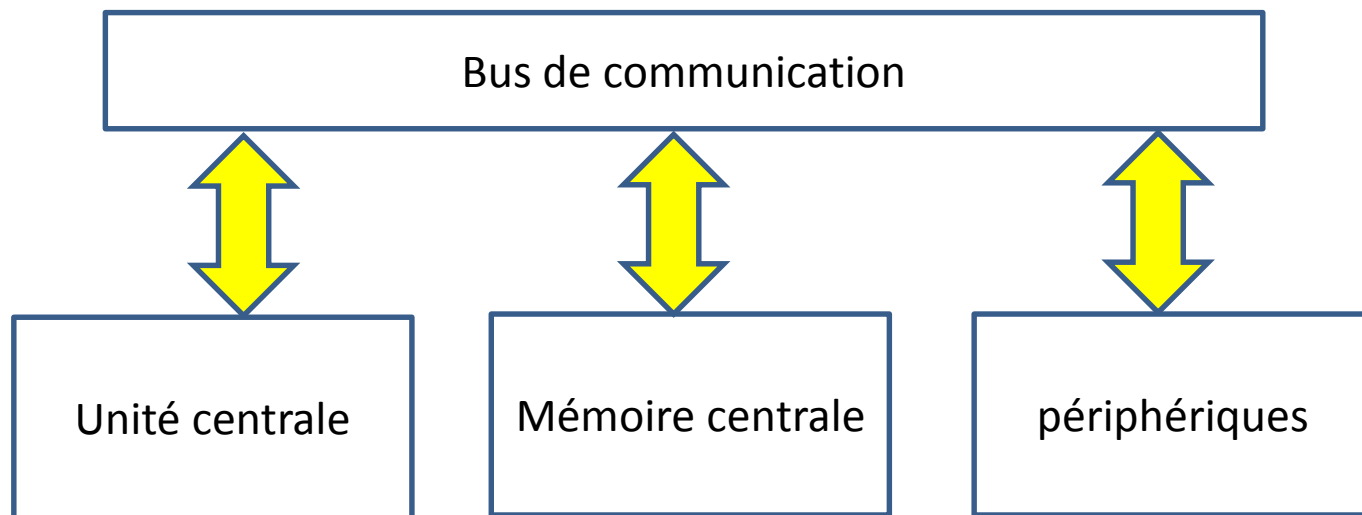
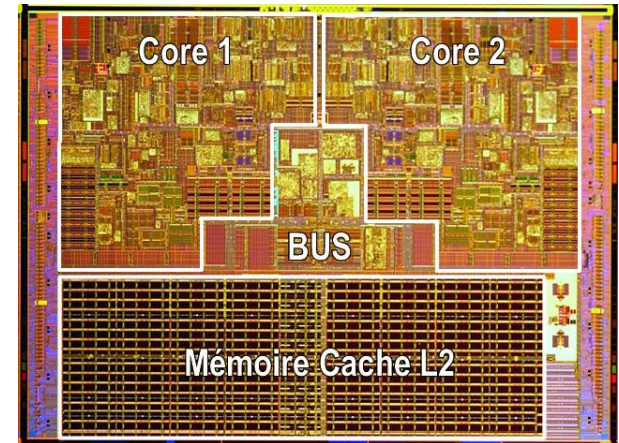
année	transistors	processeur
1971	2,300	Intel 4004, premier microprocesseur
1978	29,000	Intel 8086, premiers PC
1979	68,000	Motorola 68000
1989	1,180,000	Intel 80486
1993	3,100,000	Pentium
1997	9,500,000	Pentium III
2000	42,000,000	Pentium 4
2012	1,400,000,000	Quad-Core + GPU Core i7
2012	5,000,000,000	62-Core Xeon Phi
2014	5,560,000,000	18-core Xeon Haswell (photo)
2015	7,100,000,000	IBM z13 Storage Controller

Source : http://en.wikipedia.org/wiki/Transistor_count

Architecture de Von Neuman

Un ordinateur comporte:

- **une unité centrale (UC)**
- **des mémoires (contenant données et programmes)**
- **des périphériques**



Architecture de Von Neuman

La Mémoire Centrale

- **Stockage des données et des programmes**
- **codés par des suites de 0 et de 1**
- **Chaque cellule mémoire est désignée par son adresse**
- **Toutes les cellules ont la même taille (mot), exprimée en nombre de bits ou d'octets.**

Opérations sur une cellule:

lecture du contenu *Adresse*  *Valeur*
écriture d'une *Valeur* à une *Adresse*

Architecture de Von Neuman

Les périphériques

Claviers, écrans, souris, crayons optiques, lecteurs de codes à barre, capteurs, synthétiseurs vocaux / musicaux, lecteurs de disquettes, numériseurs (scanners), modems, imprimantes, unités de disques, bandes magnétiques, disques optiques, traceurs, réseaux, tablettes de projection, etc.

Classification possible :

périphériques d'entrée

périphériques de sortie

périphériques de stockage

2.Codage(s) de l'information

- Utilisation
 - le stockage en mémoire
 - la manipulation des données
 - la communication avec les périphériques
- Architecture de Von Neumann

2.Codage(s) de l'information

Quelques rappels:

$$\underbrace{2 \times 2 \times 2 \dots \times 2}_{\substack{\text{a fois} \\ \text{a fois}}} = 2^a$$
$$\frac{\underbrace{2 \times 2 \times \cancel{2} \dots \times \cancel{2}}_{\text{a fois}}}{\underbrace{\cancel{2} \times \cancel{2} \times \cancel{2} \dots \times \cancel{2}}_{\text{b fois}}} = 2^{a-b}$$

a > b dans cet exemple

2.Codage(s) de l'information

$$\frac{\underbrace{2 \times 2 \times 2 \dots \times 2}_{a \text{ fois}}}{\underbrace{2 \times 2 \times 2 \dots \times 2}_{a \text{ fois}}} = 1 = 2^{a-a} = 2^0$$

$$\frac{1}{2 \times 2 \times 2 \dots \times 2} = 2^{-a}$$

$$\frac{1}{2} = 2^{-1}$$

2.Codage(s) de l'information

Un octet : $2^8 = 256$ valeurs différentes

Codage des nombres en binaire non-signé : sur un octet, nombres de 0 à 255

contenu	1	0	1	0	0	0	1	1	
Poids	128	64	32	16	8	4	2	1	
		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
total = 163		128		32				2	1

contenu	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
Poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$\text{total} = c_7 \cdot 2^7 + c_6 \cdot 2^6 + c_5 \cdot 2^5 + c_4 \cdot 2^4 + c_3 \cdot 2^3 + c_2 \cdot 2^2 + c_1 \cdot 2^1 + c_0 \cdot 2^0$$

2.Codage(s) de l'information

Nombre à virgule

contenu	1	,	1		1	0	1	
Poids	1		0.5	0.25	0.125	0.0625		
			2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	
total = 1.8125	1	0.5	0.25		0.0625			

Ces nombres sont codés par « la notation virgule flottante » simple précision, double précision ou précision étendue.

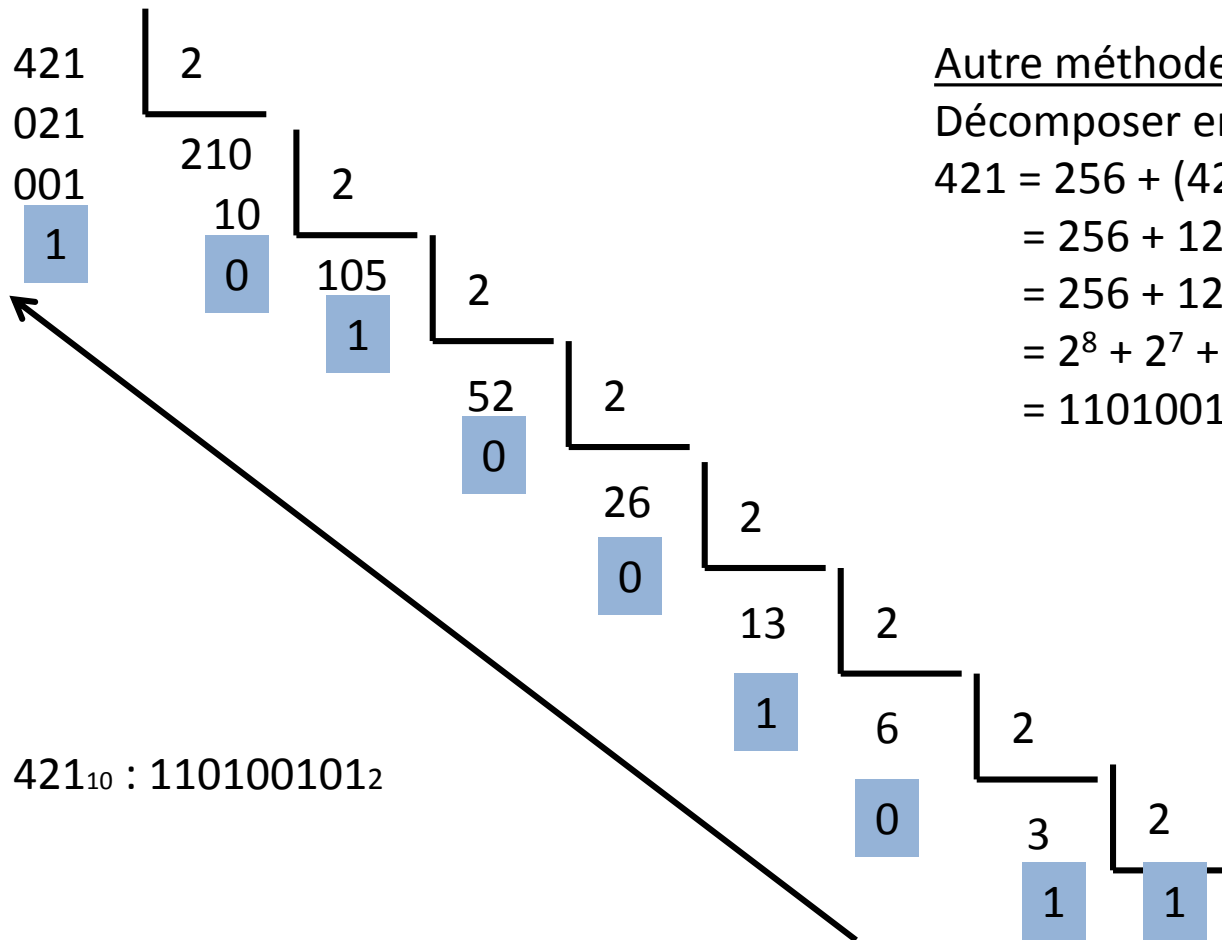
- Nombre en base n : est composé de chiffres $< n$

contenu	c_3	c_2	c_1	c_0	,	c_{-1}	c_{-2}	c_{-3}	c_{-4}
Poids	n^3	n^2	n^1	n^0		n^{-1}	n^{-2}	n^{-3}	n^{-4}
total =	$c_3 \cdot n^3$	$+ c_2 \cdot n^2$	$+ c_1 \cdot n^1$	$+ c_0 \cdot n^0$		$+ c_{-1} \cdot n^{-1}$	$+ c_{-2} \cdot n^{-2}$	$+ c_{-3} \cdot n^{-3}$	$+ c_{-4} \cdot n^{-4}$

- Nombre en hexadécimal : base 16 (chiffres < 16 de 0 à 15)
 - raccourci de l'écriture d'un quartet en binaire
 - 0 (0000), 1 (0001), 2 (0010), 3 (0011), 4 (0100), 5 (0101), 6 (0110), 7 (0111), 8 (1000), 9 (1001), A (1010), B (1011), C (1100), D (1101), E (1110) et F (1111)

2.Codage(s) de l'information

Passage d'un nombre décimal en un nombre en base n
Méthode des divisions successives pour la partie entière
Exemple: convertir 421 en base 2



Autre méthode pour le binaire:

Décomposer en puissance de 2

$$\begin{aligned} 421 &= 256 + (421-256) \\ &= 256 + 128 + (165-128) = 256 + 128 + 37 \\ &= 256 + 128 + 32 + 4 + 1 \\ &= 2^8 + 2^7 + 2^5 + 2^2 + 2^0 \\ &= 110100101_2 \end{aligned}$$

2.Codage(s) de l'information

Passage d'un nombre décimal en un nombre en base n

Méthode des multiplications successives pour la partie décimale

Exemple: convertir 0,67 en base 2

$0,67 = 0,a_0a_1a_2a_3\dots$ où $a_i = 0$ ou 1

$2 \times 0,67 = 1,34 = a_0,a_1a_2a_3\dots \Rightarrow a_0 = 1$

$0,34 = 0,a_1a_2a_3\dots$

$2 \times 0,34 = 0,68 = a_1,a_2a_3a_4\dots \Rightarrow a_1 = 0$

$2 \times 0,68 = 1,36 = a_2,a_3a_4\dots \Rightarrow a_2 = 1$

Etc...

$0,67 = 0,10101011\dots$

$$\begin{array}{r} 0,67 \\ \times 2 \\ \hline 1,34 \\ \times 2 \\ \hline 0,68 \\ \times 2 \\ \hline 1,36 \\ \times 2 \\ \hline 0,72 \\ \times 2 \\ \hline 1,44 \\ \times 2 \\ \hline 0,88 \end{array}$$

On enlève 1
avant multiplication

2.Codage(s) de l'information

Conversion d'un nombre binaire en octal (base 8) et en hexadécimal (base 16)

Octal

Puissance de 8

$8 = 2^3$ groupe de 3

$2^6 = 8^2$

$2^3 = 8$



			←			→								
1	0	0	1	1	0	1	1	1	1	1	1	0	1	0
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
4	8 ²		6	8 ¹		7	8 ⁰		7	8 ⁻¹		2	8 ⁻²	

$100110111,11101_2 = 467,72_8$

Hexadécimal

Puissance de 16

$16 = 2^4$ groupe de 4

																			
0	0	0	1	0	0	1	1	0	1	1	1	0	1	0	0	0			
2^8				2^7				2^6				2^5				2^4			
2^3				2^2				2^1				2^0				2^{-1}			
2^{-2}				2^{-3}				2^{-4}				2^{-5}							
2	3	2	2	2	1	2	2	0	2	3	2	2	1	2	0	2			
1				16^2				3				16				7			
E				16^{-1}				8				16^{-2}							

$100110111,11101_2 = 137,E8_{16}$

2.Codage(s) de l'information

Codage en complément à deux (excédent 256).

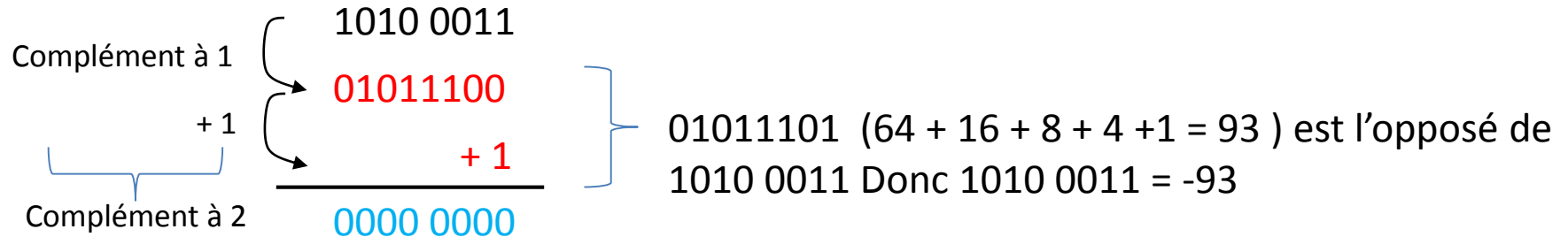
Sur un **octet**, codage de -128 à +127 :

1010 0011

163 - 256 = -93

a et b sont opposés si et seulement si $a+b=0$. Quel est l'opposé de 1010 0011?

(Rappel : $1111\ 1111 + 1 = 1\ 0000\ 0000 = 0000\ 0000$ sur 8 bits)



-128	-127	-1	0	1	127	128
1000 0000	1000 0001	1111 1111	0000 0000	0000 0001	0111 1111	0 1000 0000

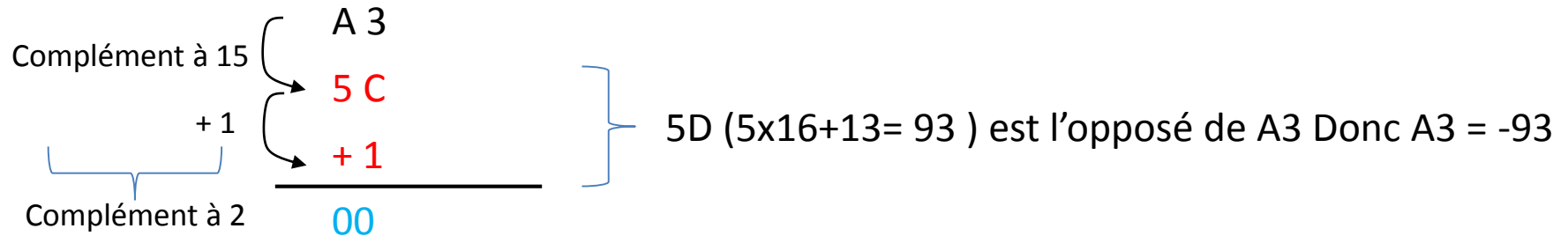
Le bit de poids fort (pF) donne une indication sur le signe: 1 négatif et 0 positif sur **un nombre de bits fixé** ici 8

2.Codage(s) de l'information

Codage en complément à deux et notation hexadécimale.

soit XY un nombre en hexadécimal représentant deux quartets soit un octet

Le nombre est négatif si le bit de poids fort du quartet de poids fort est à 1 (au moins 1000_2) soit si ce quartet X est supérieur ou égal à 8.



-128	-127	-1	0	1	127
80	81	FF	00	01	7F

Codage(s) des caractères : Code ASCII

Préambule: Pour représenter des caractères dans un fichier texte, on associe une séquence de bits (code) à une lettre, un chiffre ou un symbole.

Le premier codage largement répandu est l'ASCII (American Standard Code for Information Interchange) créé en 1967:

- Code de 7 bits (0 à 127 (2^7)) caractères anglais, nombres de 0 à 9 et certains caractères spéciaux.

- Il ne définit pas les codes de 128 à 255 (pas de lettres accentuées)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Codage(s) des caractères : ISO (codes ASCII étendus)

ISO (International Organization for Standardisation) a créé des standards pour les codes de 128 à 255, le plus connu :

ISO – 8859 – 1 (Latin 1) inclue les langues européennes

ISO-8859-1																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	ı	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Codage(s) des caractères :

Codage universel : Unicode

Unicode: chaque symbole appelé point de code reçoit un nom officiel (par ex « lettre majuscule latine c cédille ») et un index (U+00C7 codé en hexadécimal)

Les points sont regroupés par plages:

- Le latin de base (correspond à ASCII) de 00 à 7F
- Le supplément latin 1 (lettres accentuées Europe de l'ouest) de 80 à 8F
- l'alphabet phonétique international de 250 à 2AF
- les symboles musicaux byzantins de 1D000 à 1D0FF...

UTF-32

Chaque index est codé sur 32 bits. C cédille : 00 00 00 C7

Inconvénient : 4 octets par caractère alors que pour la plupart des caractères utilisés 8 bits suffisent.

Codage(s) des caractères :

Codage universel : Unicode

UTF-8 : un codage à taille variable

Les caractères les plus utilisés de 0 à 127 sont codés sur un octet, les caractères de 128 à 1023 sur 2 octets ...

Le nombre de 1 en bits de poids fort indique le nombre d'octets utilisés

Représentation	Signification
0xxx xxxx	1 octet codant 1 à 7 bits
110x xxxx 10xx xxxx	2 octets codant 8 à 11 bits
1110 xxxx 10xx xxxx 10xx xxxx	3 octets codant 12 à 16 bits
1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx	4 octets codant 17 à 21 bits

Remarques:

- Un texte écrit en ASCII (US) reste inchangé
- Un texte écrit en ISO-8859-1 est modifié pour les lettres accentuées (en UTF-8 sur 2 octets)
- Certaines opérations, comme chercher le nième caractère nécessitent un parcours séquentiel, car on ne peut pas prédire la taille occupée par les n-1 caractères précédents

2. Codage(s) de l'information: Virgule flottante

L'intervalle des nombres utilisés dans un calcul peut être très grand:

masse d'un électron: $9 \times 10^{-28} \text{g}$

masse du soleil: $2 \cdot 10^{33} \text{g}$

Donc si on veut conserver les 34 chiffres avant la virgule de l'un et les 28 chiffres après la virgule de l'autre il faudrait donc 62 chiffres significatifs et la plupart de ces chiffres seraient des 0.

Il faut donc créer un système de représentation des nombres dans lequel l'intervalle des nombres exprimables soit indépendant du nombre de chiffres avant ou après la virgule

2. Codage(s) de l'information: Virgule flottante

Notation en virgule flottante

Exprimer les nombres à l'aide de la notation scientifique:

$$n = f \cdot 10^e$$

f: mantisse

e: nombre entier positif ou négatif, appelé l'exposant

La version informatique de cette notation est l'expression en virgule flottante.

La dimension de l'intervalle est exprimée par l'exposant et la précision par le nombre de chiffre de la mantisse.

C'est une variante de cette représentation qui est utilisée pour les ordinateurs

2. Codage(s) de l'information: Virgule flottante

Pour des raisons d'efficacité, on utilise les bases 2, 4, 8, ou 16 plutôt que 10.

Normalisation

Si le chiffre de la mantisse situé le plus à gauche est à 0, on décale tous les chiffres d'une position vers la gauche et on décrémente l'exposant de 1.

Une mantisse dont le chiffre le plus à gauche est non nul est dite normalisée.

La représentation d'un nombre est alors unique.

La virgule binaire ou hexadécimale est alors supposée être immédiatement à gauche du bit de poids fort de la mantisse.

2. Codage(s) de l'information: Virgule flottante

0	100 1001	, 1101 1100 0000 0000
---	----------	-----------------------

On utilise dans ce mode de représentation la méthode de codage par excédent à 64.

La représentation flottante: IEEE 754

Un comité de l'IEEE (Institute of Electrical and Electronics Engineers, association de professionnels des secteurs de l'électronique) s'est constitué avec pour objectif de définir un standard pour les calculs arithmétiques flottants:

- permettre les échanges entre les différents appareils
- offrir une norme précise au concepteur d'ordinateurs

Résultat le standard IEEE 754:

Trois formats de représentation des nombres flottants :

la simple précision sur 32 bits la double précision sur 64 bits la précision étendue sur 80 bits

2. Codage(s) de l'information: Virgule flottante

Simple précision

Bit de signe	Exposant 8 bits	Mantisse sur 23 bits
--------------	-----------------	----------------------

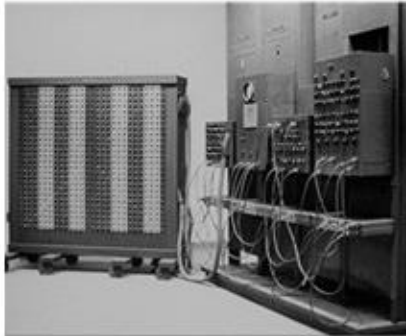
Double précision

Bit de signe	Exposant 11 bits	Mantisse sur 52 bits
--------------	------------------	----------------------

L'exposant est codé excédent à 127 pour la simple précision et en excédent à 1023 pour la double.

Une mantisse est dite normalisée quand le premier bit qui suit la virgule vaut 1. Par hypothèse du standard IEEE la mantisse est normalisée (standard), donc le premier bit est toujours égal à 1, on peut donc s'en passer et noter implicitement sa présence. Ainsi une mantisse IEEE comprend un bit supposé à 1 qu'on appelle bit caché, puis 23 ou 52 bits de valeur quelconque, de même la virgule est implicite après le bit caché. Donc la mantisse est appelée dans le standard IEEE pseudomantisse ou significande.

3. Systèmes d'exploitation



The ENIAC Today



www.old-computers.com



3.1 Introduction aux Systèmes d'exploitation

Contenu

- à quoi ça sert
- grandes fonctions
- systèmes existants
- ce que vous allez utiliser

A quoi ça sert ?

Un système d'exploitation sert à **exploiter** une machine

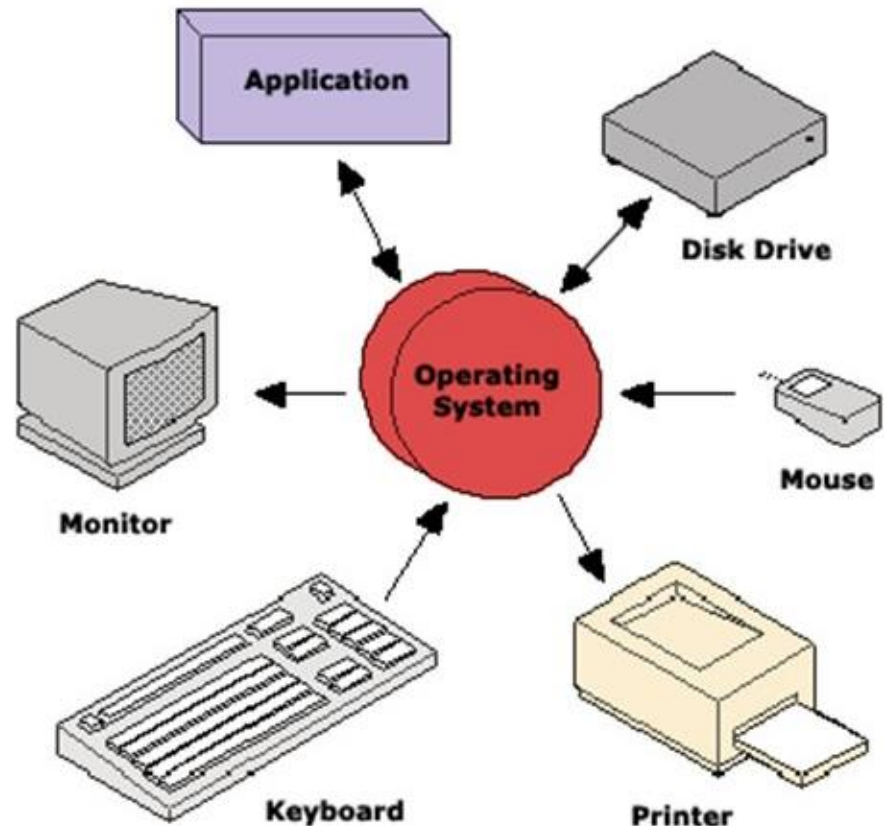


(Operating System)

Définition

Système d'exploitation :
ensemble de programmes
qui assurent la liaison
entre

- les ressources matérielles
- les applications



Fonctions d'un système d'exploitation

Dirige le fonctionnement de l'ordinateur

- initialise le matériel
- fonctions de base pour piloter les périphériques
- gestion des fichiers
- assure la gestion, l'ordonnancement et la communication des processus (tâches)
- ...

On en trouve partout



Le marché

Il existe des centaines de systèmes :

- Windows,
- Unix : Linux, Solaris, AIX, HP-UX, FreeBSD ...
- GCOS, VMS, AS400 ...
- ...

Inconnus du grand public :

- mainframes
- systèmes embarqués

Au département informatique

Vous utiliserez principalement

- Linux (Debian 8 “Jessy”)
- Windows 10



3.2 Le cours de systèmes d'exploitation

Contenu

- Objectifs du cours

Objectifs du cours

- Notions sur les systèmes d'exploitation
- Connaître l'environnement local
- Utilisation concrète des Systèmes Informatiques (USI)
- Utilisation du langage de commande
 - Commandes de base
 - Filtres
 - Programmation des scripts

3.3 Environnement de travail

L'environnement de travail du département informatique de l'IUT de Bordeaux

Contenu

- les utilisateurs
- le réseau du département
- comment se connecter
- principales adresses

Les utilisateurs

- environ 250 étudiants (11 groupes)
 - DUT A1 = 110, A2 = 80
 - DUT AS = 20
 - LP = 20 + 20 + 15
- une centaine d'enseignants
 - permanents
 - extérieurs
 - vacataires et professionnels

Le réseau du département (Bordeaux)

Réseau **filaire** (ethernet) et **Wifi** reliant

- une centaine de **stations**, dual boot Linux + Windows (salles en libre service)
- une douzaine de **serveurs** (fichiers partagés, services)
- des **imprimantes** (une par salle)
- ...



Connexions

Le département est relié au reste du monde via

- **REAUMUR** : REseau Aquitain des Utilisateurs des Milieux Universitaires et de la Recherche
- **RENATER** : REseau NATional pour l'Enseignement et la Recherche

Utilisation des équipements limitée aux usages pédagogiques (charte REAUMUR).



Comptes Université de Bordeaux

Chaque usager de l'Université de Bordeaux possède un **compte** qui donne accès

- à une **boîte aux lettres**
- à un ENT environnement numérique de travail (ent.u-bordeaux.fr)
- à des services

Important

Chaque utilisateur est **responsable** de son compte, qui est protégé par un **mot de passe**

Comptes Département Informatique

Les usagers du **département informatique** ont, de plus, accès à des postes de travail et des serveurs avec

- des fichiers
- des logiciels
- des services

L'authentification est faite par les serveurs de l'Université de Bordeaux (même nom d'utilisateur, même mot de passe)

Boîte aux lettres

Adresses e-mail

- **étudiants** : prenom.nom@etu.u-bordeaux.fr
- **personnels** : prenom.nom@u-bordeaux.fr

Consultation/Envoi

- par l'interface web de l'ENT <http://ent.u-bordeaux.fr>
- serveur IMAP (webmel.u-bordeaux.fr sécurisé par SSL, port 993)
- serveur SMTP (smtpauth.u-bordeaux.fr sécurisé par SSL, port 465)

Domaines

u-bordeaux.fr : l'université de Bordeaux

- ent.u-bordeaux.fr, environnement numérique de travail
- prenom.nom@etu.u-bordeaux.fr, votre adresse mel
- ...

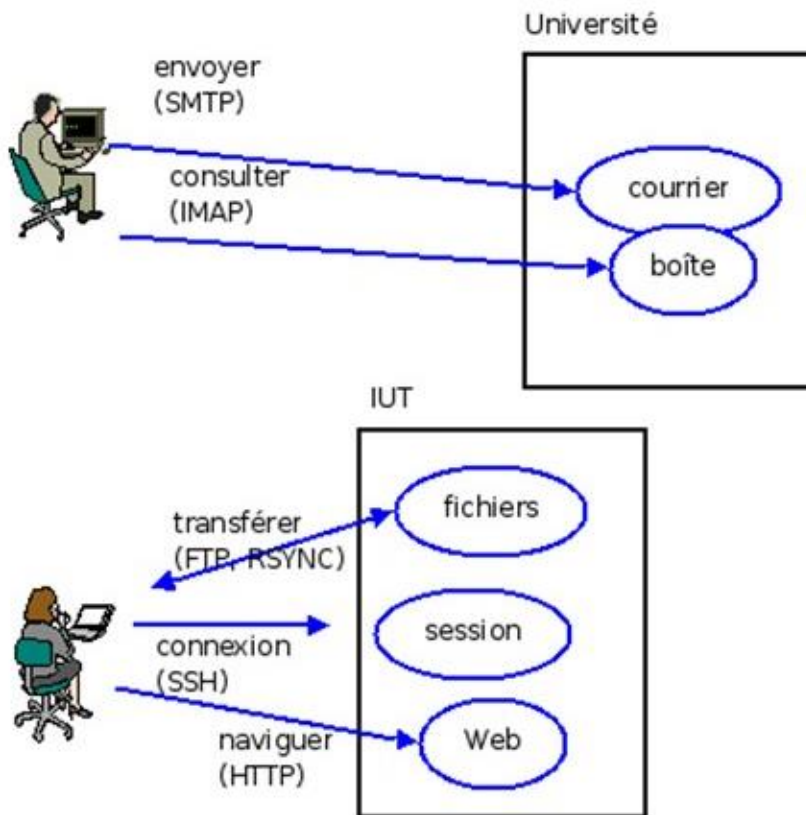
iut.u-bordeaux.fr : l'IUT

- www.iut.u-bordeaux1.fr, site web IUT
- info-ssh1.iut.u-bordeaux.fr, connexion au département
- ..

Les adresses “u-bordeaux1” sont un vestige du passé.

Les services

Les **services** accessibles depuis l'extérieur



Site officiel du département

<http://www.iut.u-bordeaux.fr/info>

The screenshot shows the official website of the IUT Bordeaux 1 Department of Informatics. The header features the department's logo on the left and a search bar on the right with the text "Rechercher dans le site : Rechercher" and a "VALIDER" button. Below the header is a navigation menu with five colored buttons: "DÉPARTEMENT INFORMATIQUE" (pink), "DUT INFORMATIQUE" (blue), "LICENCES PROFESSIONNELLES" (green), "ESPACE ENTREPRISE" (light green), and "PROJETS ÉTUDIANTS" (orange). On the left side, there are two sections: "Vous êtes" (You are) with a list of user types (Un lycéen, un étudiant, Une entreprise) and "Nous contacter" (Contact us) with the department's address. The main content area displays three featured images with captions and arrows: "DUT Informatique", "Licences professionnelles", and "Espace entreprises". At the bottom, the word "ACTUALITÉS" (News) is visible.

Département Informatique IUT BORDEAUX 1

Rechercher dans le site : Rechercher VALIDER

DÉPARTEMENT INFORMATIQUE DUT INFORMATIQUE LICENCES PROFESSIONNELLES ESPACE ENTREPRISE PROJETS ÉTUDIANTS

Vous êtes

- Un lycéen
- un étudiant
- Une entreprise

Nous contacter

Département informatique
IUT Bordeaux I,
15, rue Naudet
CS 10207
33175 Gradignan Cedex
France

DUT Informatique

Licences professionnelles

Espace entreprises

ACTUALITÉS

(informations publiques)

Site pédagogique

<https://intranet.iut.u-bordeaux.fr/info/pedago/>



(usagers du département)

Le domaine iut.bx

Adresses privées du réseau local de l'IUT. A usage interne.

- stations de travail
- serveurs
- imprimantes
- ...

4. Utiliser Linux

Contenu

- c'est quoi Linux ?
- comment se connecter
- le bureau GNOME

Linux en deux mots

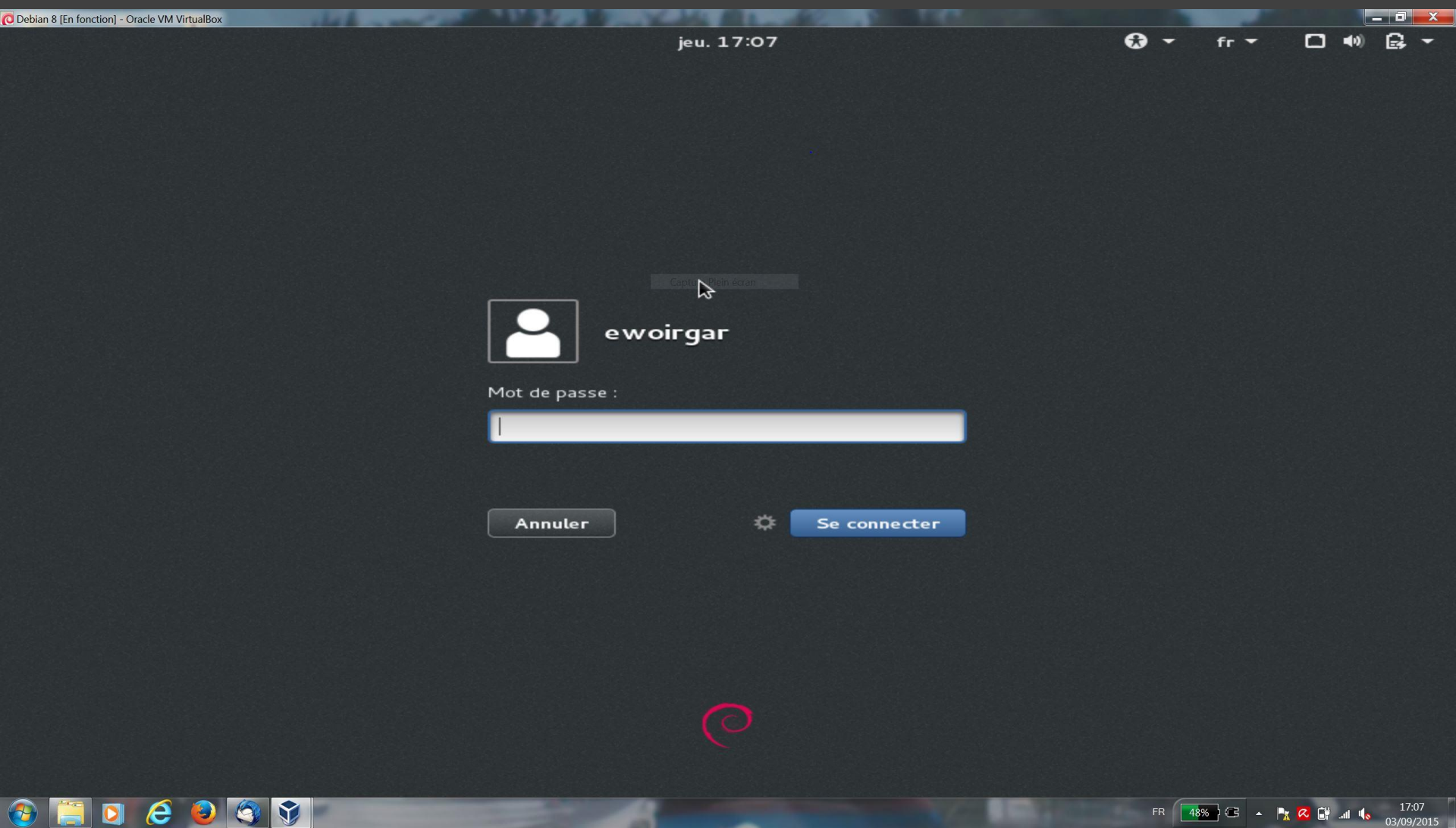
Linux : système d'exploitation

- créé en **1992** par Linus Torvalds
- Membre de la famille **Unix** (1969) : système généraliste, multi-tâche, multi-utilisateurs, partages de ressources, réseau. Conforme à la norme **POSIX**.
- En réalité Linux est le nom d'un **noyau** utilisé dans de nombreuses **distributions** : Ubuntu, Mandriva, Debian, Gentoo, Slackware, etc.

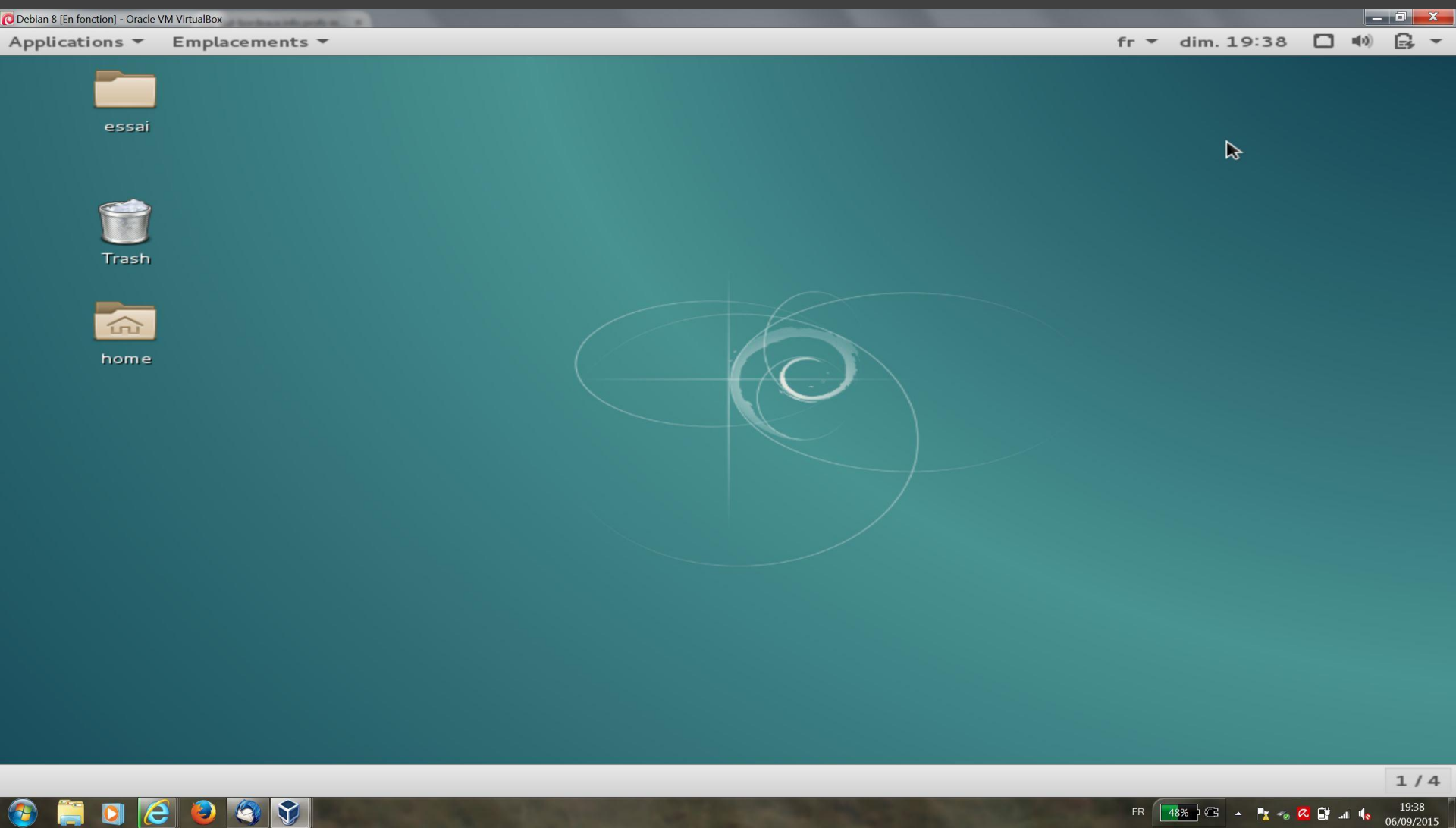
Ici, distribution Debian 8 “Jessy”



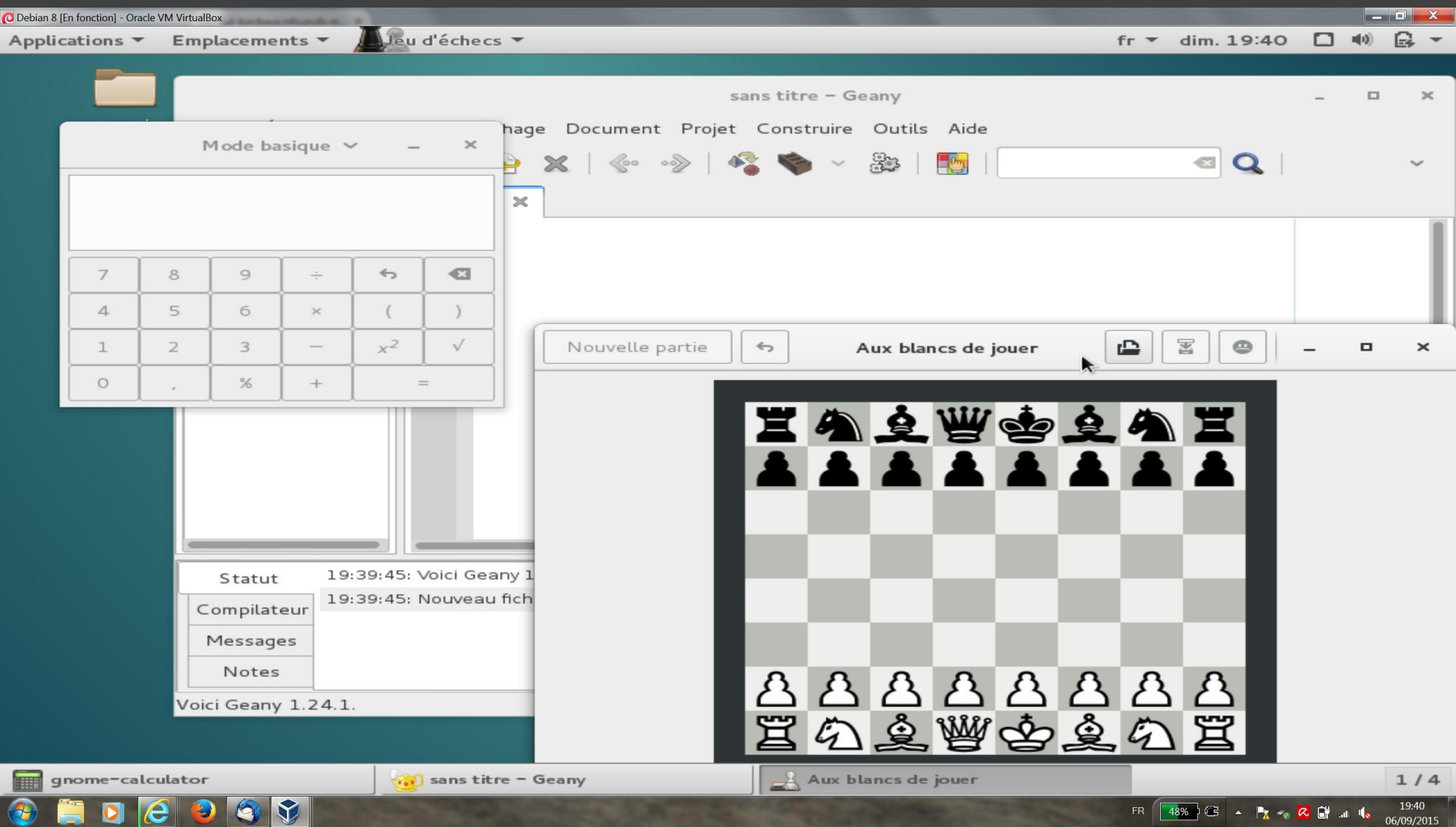
Connexion locale (bannière graphique)



Le bureau GNOME (classic)



Fenêtres



Interaction graphique

En mode graphique, interaction basée sur

- menus, sous-menus
- fenêtres, icones, barre de titres
- pointeurs, curseurs
- souris (clics, déplacements), modificateurs
- clavier
- ...

3.5 Lignes de commandes

Contenu

- principe
- syntaxe
- exemple
- où trouver la documentation?

Interface lignes de commandes

Les "Interprètes de ligne de commande" (**shells**) permettent de lancer des actions décrites par des lignes de commandes.

Exemples :

- `ls`
- `cp /mnt/usbdisk/*.mp3 Ma_Musique`
- `ls Ma_Musique | mpage -4 -Pbureau421`

Interface graphique ou texte ?

- interfaces graphiques : plus accessibles aux utilisateurs occasionnels
- ligne de commande : **usage plus efficace**, nécessitant un investissement (rentable) pour les utilisateurs professionnels.
Permet à la fois
 - usage interactif
 - exécution de **scripts** (suites de commandes enregistrées).
automatisation de tâches répétitives.

Complémentarité : un script peut être lancé par une interface graphique, et inversement.

Syntaxe des commandes

Une commande comporte

- le **nom d'un programme**
- des **options** (précédées par un tiret)
- des **paramètres**

Ces éléments sont séparés par des espaces.

Exemple

```
a2ps -r -Pbureau421 premier.cc doc.txt
```

Exemple de commande

Exemple

```
a2ps -r -Pbureau421 premier.cc doc.txt
```

Détails expliqués

- **a2ps** : programme de formattage et impression
- **-r** : orientation "paysage"
- **-Pbureau421** : sur l'imprimante bureau421
- **premier.cc doc.txt** : noms des fichiers à imprimer

Où trouver de la documentation

Documentation intégrée (option -h ou --help)

```
$ cp --help
```

```
Usage: cp [OPTION]... [-T] SOURCE CIBLE
```

```
    ou: cp [OPTION]... SOURCE... RÉPERTOIRE
```

```
    ou: cp [OPTION]... --target-directory=RÉPERTOIRE SOURCE...
```

Copier la SOURCE vers la DESTINATION, ou de multiples SOURCES vers un RÉPERTOIRE.

Les arguments obligatoires pour les options de formes longues

le sont aussi pour les options de formes courtes.

-a, --archive	identique à -dpR
--backup[=CONTROLE]	archiver chaque fichier de destination
-b	identique à --backup mais sans argument
--copy-contents	copier le contenu des fichier spéciaux
	en mode récursif
-d	identique à --no-dereference --preserve=link

Où trouver de la documentation (suite)

Pages de manuel (commande **man**)

```
$ man cp
```

CP(1)

User Commands

CP(1)

NAME

cp - copy files and directories

SYNOPSIS

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

DESCRIPTION

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.

-a, --archive
same as -dpR

--backup[=CONTROL]
make a backup of each existing destination file

Où trouver de la documentation (suite 2)

Autres sources de documentation

- commande info
- documentations installées (/usr/share/doc)
- sites pédagogiques
- forums d'aide
- moteurs de recherche
- êtres humains
- ...

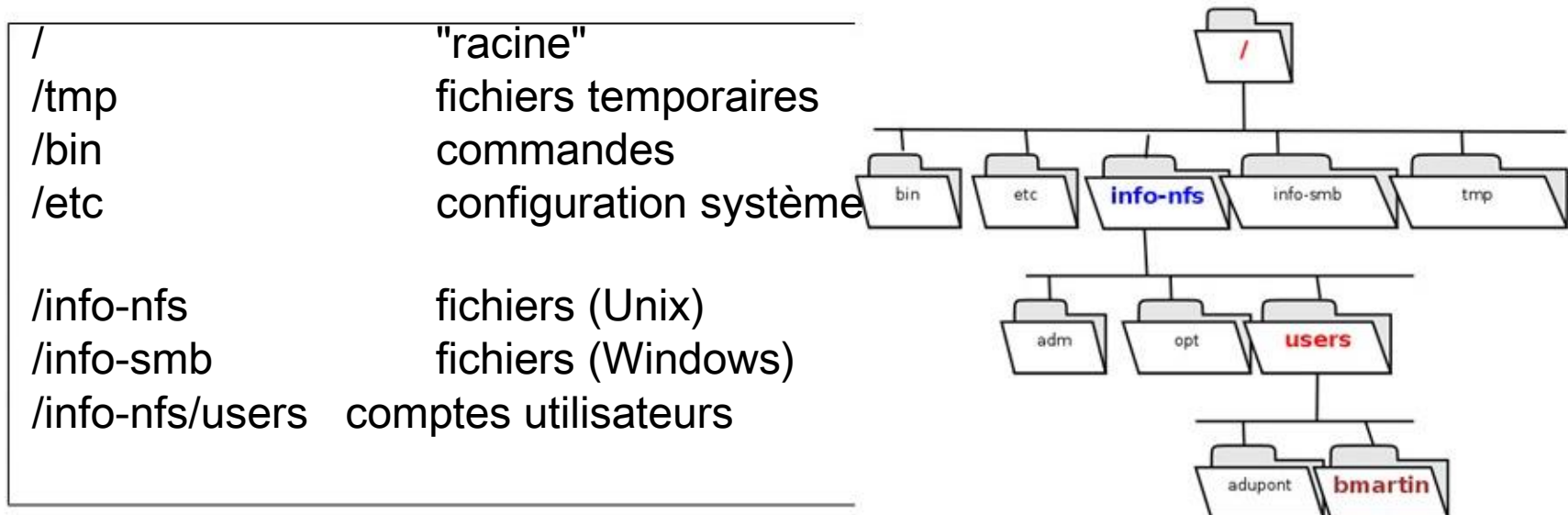
3.6 Arborescence

Contenu

- arborescence = fichiers + répertoires
- chemins d'accès relatifs et absolus
- répertoire de travail

Arborescence des fichiers et répertoires

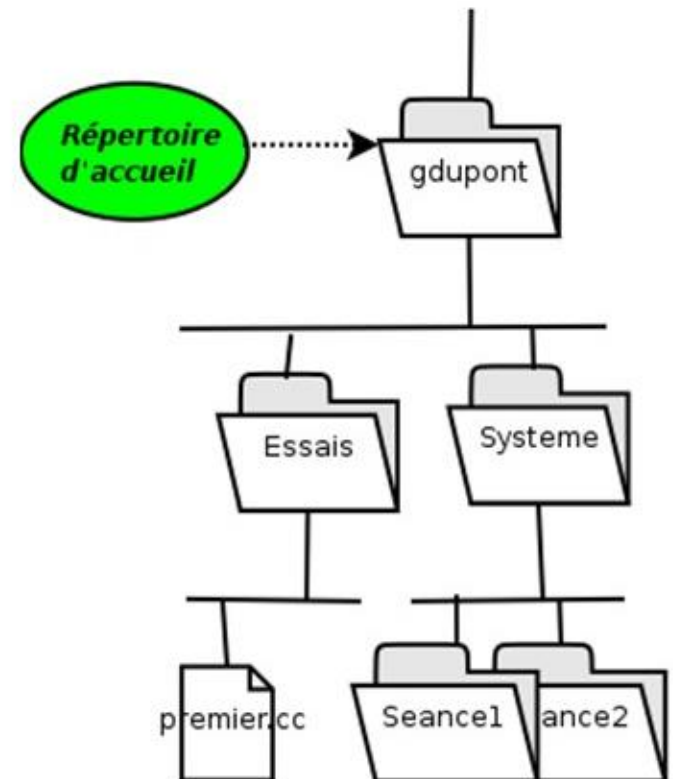
Les données et les programmes sont stockés dans une **arborescence** de fichiers et répertoires.



Répertoire d'accueil

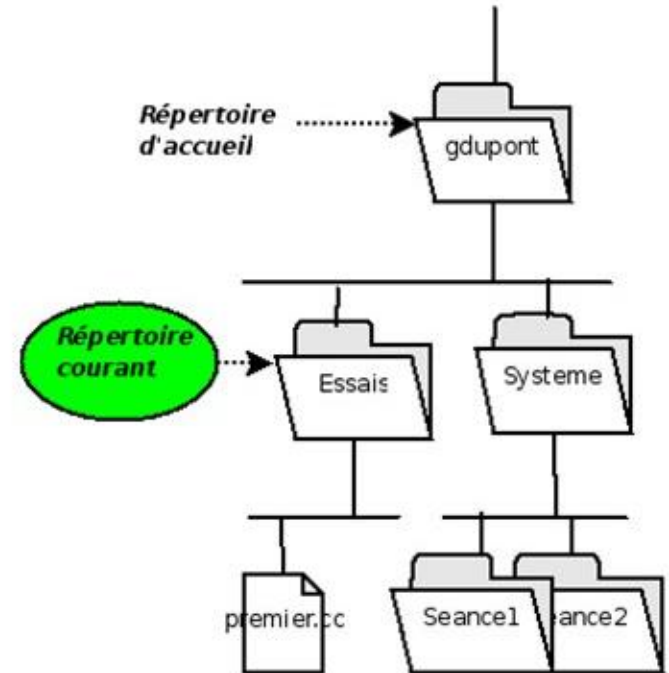
Le répertoire d'accueil d'un utilisateur

- hébergé sur un serveur commun
- partagé entre toutes les machines du département
- protégé par des droits d'accès



Chemins d'accès relatif

- **Chemin d'accès** : désignation de la **position** d'un fichier (ou d'un répertoire) dans l'arborescence
- **Chemin relatif** : en fonction du **répertoire courant**
- Exemples :
 - premier.cc,
 - ..
 - ../Systeme/Seance2



Chemins d'accès absolus

Chemin d'accès **absolu** : indépendant du répertoire courant.

Exemples

- `/info-nfs/users/gdupont/Essais/premier.cc`
- `~gdupont/Essais/premier.cc`
- `~/Essais/premier.cc`

Exemple

La commande

```
cp /mnt/usbdisk/*.mp3 Ma_Musique
```

copie des fichiers (commande cp = copy)

- **Quoi ?** Tous les fichiers
 - du répertoire /mnt/usbdisk (chemin absolu)
 - dont le nom se termine par .mp3
- **Où?** Dans le sous-répertoire Ma_Musique du répertoire courant (chemin relatif)

3.7 Commandes utiles

Contenu

- créer/supprimer/déplacer des fichiers, des répertoires
- développer des programmes
- éditeur geany

Premières commandes

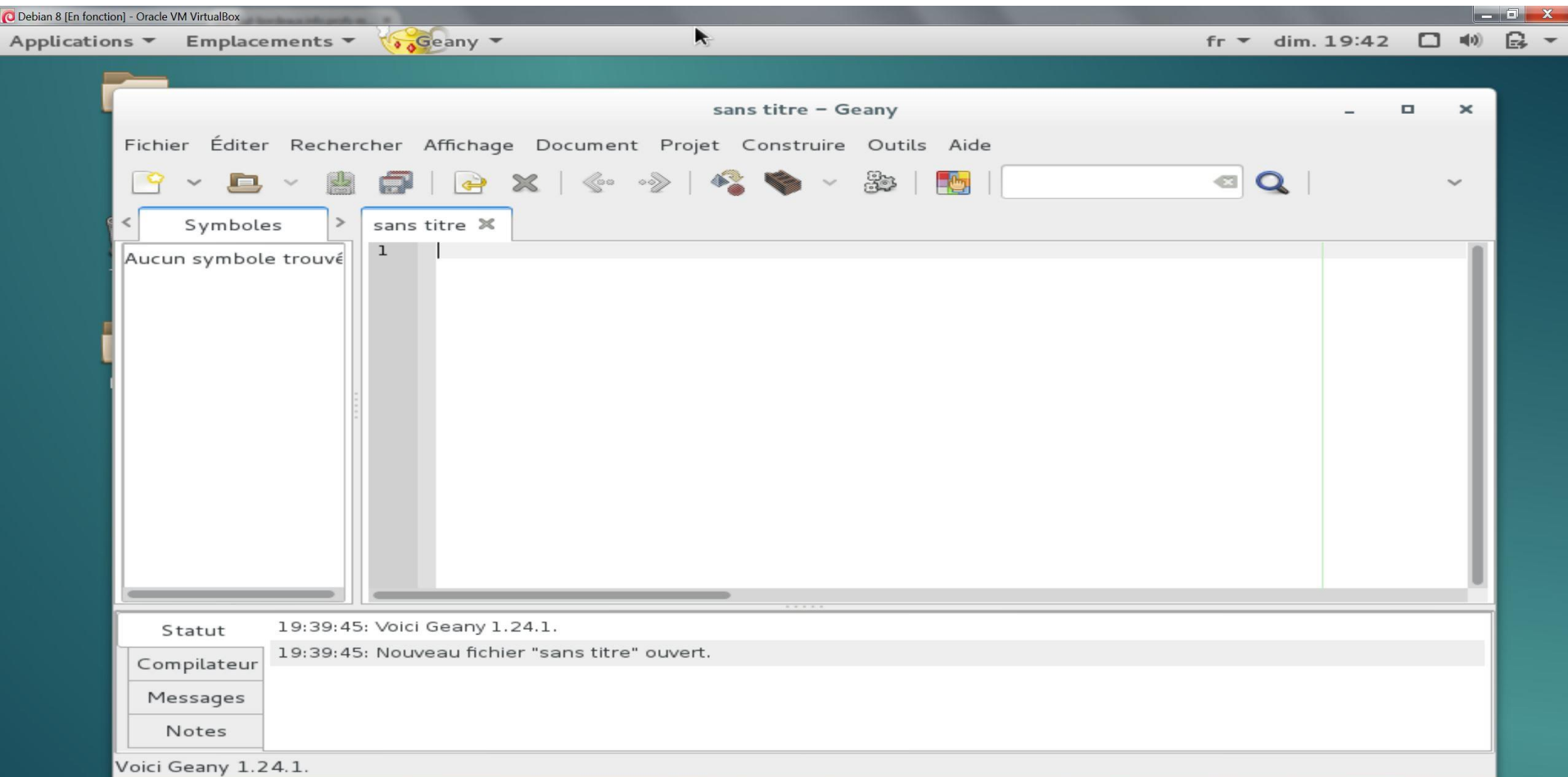
mkdir	répertoire	crée un répertoire (make directory)
rmdir	répertoire	supprime (remove) un répertoire
cd	répertoire	change répertoire courant
pwd		affiche (print) chemin du rép. courant
ls		affiche (list) contenu du rép. courant
cp	fichier... destination	copie (copy) fichiers
mv	fichier... destination	déplace (move) fichiers
rm	fichier...	supprime fichiers
cp -r	répertoire... destination	copie récursive d'un répertoire
rm -r	répertoire...	suppression récursive d'un répertoire

Objectif à court terme

- commandes de base
 - ...
- développement de programmes
 - édition de texte : pour taper le texte source, le modifier etc.
 - compilation : traduire le source en binaire exécutable
 - exécution

Utilisation de l'IDE Geany

IDE : Integrated Development Environment



Fonctions de l'IDE

Un IDE permet d'enchaîner rapidement 3 fonctions principales

- **Édition de textes** :
modifier le texte source du programme
- **compilation** :
commande `g++ -o essai essai.cc`
- **exécution** :
commande `./essai`

Choix IDE

Il existe de nombreux environnements de développement intégrés (IDE) : **Emacs**, **Eclipse**, ...

Pourquoi Geany ?

- léger, rapide
- multi-langage (C++, Java, PHP, ...)
- simple à utiliser
- convient pour initiation à la programmation

Les IDE plus "professionnels", comme **Eclipse** intègrent des fonctionnalités supplémentaires :

- gestion des versions,
- "refactoring" de code,
- etc

Inconvénients :

- outils "lourds"
- prise en main plus longue.

Utilisation des commandes

En cliquant sur Applications/Accessoires/Terminal on ouvre une fenêtre qui permet de taper des commandes.

Exemple:

- Exécution de *date*
- Exécution *echo « hello »*
- Exécution *geany*

Dans la fenêtre s'exécute un programme appelé interprète de commandes:

- Lire une ligne de commande
- L'analyser
- L'exécuter
- Recommencer (sauf si *exit*)

Une ligne de commande commence par le nom de la commande suivi par des options et des paramètres.

- *echo -n « hello world »*
- *date -d « Sunday »*
- *ls -l -t -r (ls -ltr)*

Aide: *man*

Historique et complétion

L'interprète de commande bash possède un **mécanisme d'historique** (flèches h/b, ! suivi des premières lettres, ~/.bash_history ...)

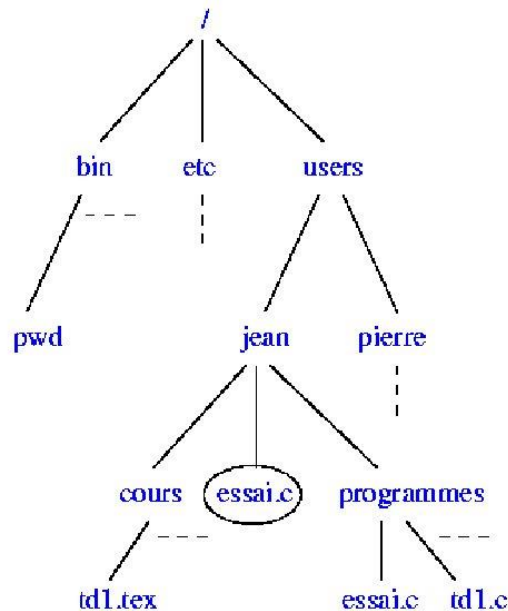
Complétion: si on tape le début du nom d'une commande, le nom ou les noms qui correspondent seront automatiquement complétés

Le principe est le même pour les noms de fichier

Navigation dans les répertoires (dossier)

Espace de travail organisé en répertoires et sous-répertoires

On appelle répertoire courant (working directory) le répertoire où on se trouve à un moment donné



Quelques commandes:

- `pwd`
- `ls` (voir les options utiles)
- `cd [DIR]`
- `mkdir [NOM]`
- `rmdir [NOM]`

Manipulation fichiers et répertoires

mkdir REP : créer un répertoire

rmdir REP : supprimer un répertoire (vide)

cp [-rv] SRC ... DST : copier

rm [-ri] F : suppression (remove)

mv F ... DST : renommer (move)

touch F ... : créer un fichier vide ou changer sa date d'accès

Créez un sous-répertoire essais dans le répertoire /tmp et essayez les commandes.
Comment déplacer un répertoires, le copier, le supprimer?

Regardez le contenu d'un fichier:

cat [-n] FICHIER

less FICHIER

more FICHIER

Méta-caractères

`cp *.cc /tmp/Essais`

* est un méta-caractère (rôle spécial) *.cc est l'ensemble des fichiers, dont le nom se termine par .cc.

* : n'importe quelle chaîne de caractères

? : n'importe quel caractère

[] : un ensemble de caractères possibles – dans [] définit un intervalle ! inverse

Exercice:

Dans /info-nfs/users, trouvez

tous les identifiants d'utilisateurs commençant par un x

tous les identifiants d'utilisateurs contenant un x

tous les identifiants d'utilisateurs contenant un x, un w ou un z

tous les utilisateurs dont le nom commence par une lettre de A à E. On suppose qu'il n'y a qu'une seule lettre pour le prénom.

Déspécialisation

Les caractères spéciaux perdent leur caractère spécial si on les précède d'un anti slash (caractère spécial)

3.8 Compression

Compression

- Sans perte

 - Fichiers de données, sources, etc.

- Avec perte

 - Image, vidéo, son...

Dans la limite de la tolérance de l'œil / l'oreille

Les exemples suivants : sans perte

- gzip bzip2 : plus courants

Copiez fichier volumineux dans /tmp

Notez sa taille

gzip fichiervolumineux

Taille de fichiervolumineux.gz ?

Essayez zcat, zless?

Comment décompresser fichiervolumineux

3.9 Archivage

TAR = Tape ARchiver

Autrefois utilisé pour archiver sur bande magnétique

BUT : Regrouper plusieurs fichiers en un seul

Fichier de sortie = **archive**

Quelques formats : .tar .cpio .zip .7z ...

Usage :

```
tar -cf usi.tar USI
```

```
tar -tf usi.tar
```

```
tar -xf usi.tar USI/semaine1/bonjour.html
```

-c=create, -f=file, -x=extract, -t=list , -v=pour voir le déroulement

Bonne pratique : se placer au dessus du répertoire à archiver

Exercice : TP1_AS

Les commandes de codage de caractères

La commande: *od -c fichier* permet de voir le contenu d'un fichier

- c : ascii
- a : ignore le bit de poids fort
- t d1 : en décimal
- t u1 : en décimal non signé
- t x1 : en hexadécimal

Exercice:

- sous geany, tapez deux lignes accentuées, sauvées en encodage ISO 8859 1 (Documents/Définir l'encodage / Européen de l'Ouest)
- regardez ce texte avec les différentes options *od*
- comment sont représentés le sauts de ligne? les caractères accentués?

La commande : *iconv -f oldcode -t newcode entree -o sortie*

iconv -f ISO-8859-1 -t UTF-8 index.iso -o index.html

Redirection

Redirection

Les commandes produisent du texte sur leur **sortie standard**

Exemples de commandes

```
$ echo "Bonjour"
```

```
$ ls
```

```
$ date "+%H:%M"
```

Ce texte peut être **redirigé** vers un fichier

Exemples : redirections sortie standard

```
$ echo "bonjour" > message.txt
```

```
$ date "+%H:%M" >> message.txt
```

Redirections sortie standard

- Remplacement d'un fichier : 

Exemple

```
$ echo "bonjour" > message.txt
```

- Extension d'un fichier : 

Exemple

```
$ date >> fichier
```

Sortie d'erreur

- Certains messages sont produits sur la **sortie d'erreur**

Mise en évidence

```
$ g++ essai.cc > resultat.log
essai.cc:1: error: ISO C++ forbids declaration of
'example' with no type
$
```

- Redirection sortie d'erreurs 2 > 2 >>

Exemple

```
$ g++ mon-programme.cc 2> erreurs.txt
$
```

Redirection sortie d'erreur sur sortie standard

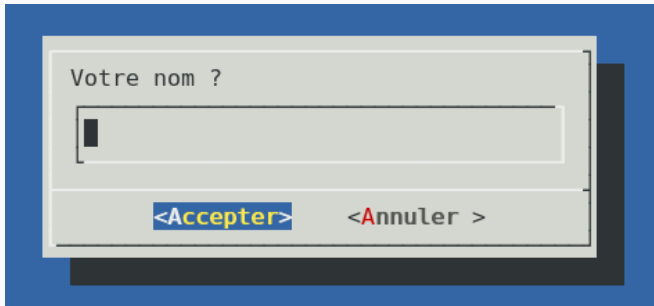
Exemple

```
$ g++ mon-programme.cc 2>&1  
essai.cc:1: error: ISO C++ forbids declaration of  
'exemple' with no type  
$
```


Autres usages de la sortie d'erreur

Exemple

```
$ dialog --inputbox "Votre nom ?" 8 40 2> /tmp/nom
```




Redirection de l'entrée standard

- Depuis un fichier : 

Exemple

```
$ tr '[a-z]' '[A-Z]' < texte.txt
```

- “here document” : 

Exemple

```
$ tr '[a-z]' '[A-Z]' <<XXX
```

```
ceci Est un
```

```
exemple
```

```
XXX
```

Redirection entre deux commandes

L'opérateur `|` (*pipe*) redirige la **sortie standard** d'une commande vers l'**entrée standard** d'une autre commande

Exemple

```
$ w | cat -n
```

On peut constituer un **pipeline** de plusieurs commandes

Exemple : redimensionner une image

```
anytopnm dscn3214.jpg |  
    pnmscale -width 100 |  
    pnmtopng > statue-hcm-100px.png
```

Les droits d'accès (bases)

Droits d'accès

La commande `ls -l` montre les **droits d'accès**

Exemple

```
billaud@feathers:~/Essais/C++$ ls -l
total 64
drwxr-xr-x 2 billaud profs 4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs 6495 nov 2 21:19 initTab
-rw-r--r-- 1 billaud profs 236 nov 2 21:19 initTab.cc
...
```

Premier caractère

- `d` pour les répertoires (*directory*)
- `-` pour les fichiers

Droits d'accès : suite

Exemple

```
drwxr-xr-x 2 billaud profs 4096 aoÿ 29 12:22 Heap
-rwxr-xr-x 1 billaud profs 6495 nov 2 21:19 initTab
-rw-r--r-- 1 billaud profs 236 nov 2 21:19 initTab.cc
```

Lettres suivantes : indiquent les **droits d'accès** Présentation par groupe de trois :

-	rwx	r-x	r-x
---	-----	-----	-----

- | |
|-----|
| rwx |
|-----|

 pour le **propriétaire** du fichier (billaud)
- | |
|-----|
| r-x |
|-----|

 les utilisateurs du **groupe profs**
- | |
|-----|
| r-x |
|-----|

 pour les autres

Les droits d'accès

Les lettres indiquent les droits d'accès (ou **mode**, ou **permissions**)

- **r** pour **read** (droit de lecture)
- **w** pour **write** (droit d'écriture, modification)
- **x** pour
 - **execute** (droit d'exécution) pour les fichiers,
 - **x=cross** (droit de traverser) pour les répertoires.

Exemples :

Exemple

```
drwxr-xr-x 2 billaud profs 4096 aoÿ 29 12:22 Heap
-rwxr-xr-x 1 billaud profs 6495 nov 2 21:19 initTab
-rw-r--r-- 1 billaud profs 236 nov 2 21:19 initTab.cc
```

- `iniTab.cc` peut être lu et modifié par son propriétaire (`rw`), lu par les membres du groupe (`r`) et par les autres (`r`).
- `a.out` peut être lu, modifié et exécuté par son propriétaire (`rw`), lu et exécuté par les utilisateurs du groupe (`rx`) ainsi que par les autres (`rx`).
- le répertoire `Heap` peut être lu, modifié et traversé par le propriétaire (`rw`), lu et traversé par les membres du groupe (`rx`) et les autres.

chmod : changer les droits d'accès

chmod : changer les droits d'accès

La commande “chmod” change les droits d'accès (CHange MODe)

Notation octale

Exemple : `chmod 750 mon-fichier`

- chaque groupe de trois bits est codé en octal, avec $r=4$, $w=2$, $x=1$.
- Donc `chmod 750 ...` donne les droits

rw	x	r	-	x	-	-
----	---	---	---	---	---	---

 au fichier

Exercices

Exercice : notation octale

Complétez la table d'équivalence

octal	droits	octal	droits
0	- - -	4	
1		5	r - x
2		6	
3		7	r w x

Exercices

Exercice : droits sur les fichiers

1. Tapez

```
$ ls -l > mon-fichier
```

```
$ chmod 777 mon-fichier
```

- pouvez-vous lire le fichier
(cat mon-fichier)?
- le modifier
(echo >> mon-fichier)?

2. même question après

```
chmod 666 mon-fichier
```

3. ...

droits	lire ?	modifier ?
000	non	non
111		
222		
333		
444		
555	oui	oui
666		
777		

Exercice : droits

1. Tapez

```
$ ls -l > mon-fichier
```

```
$ chmod 777 mon-fichier
```

- pouvez-vous lire le fichier ?
- le modifier ?

2. Changez les droits

```
$ chmod 077 mon-fichier
```

- pouvez-vous le lire ?
- le modifier ?

3. Conclusions ?

chmod : notation symbolique

Exemple

`chmod u=rwx,g=rx,o= mon-fichier`

- `u` user (propriétaire)
- `g` groupe
- `o` others (autres)

chmod : modification des droits

Sous forme symbolique, permet de **modifier** *certain*s droits.

- `+` ajouter des droits
- `-` enlever

Exemple

```
chmod go-w f
```

enlève le droit `w` au groupe (`g`) et aux autres (`o`), sans changer les autres permissions.

Exemple

```
chmod +x f
```

ajoute les droits “`x`”

Encore des commandes

Quelques commandes

- `grep` : sélection de lignes
- `cut` : sélection de colonnes
- `sort` : tri
- `join` : jointure

la commande grep

Sélectionne des lignes d'un texte, qui contiennent un certain motif (expression régulière)

exemple

- Soit le fichier `villes.txt`

```
france:paris
```

```
vietnam:ho chi minh
```

```
italie:roma
```

```
france:bordeaux
```

```
vietnam:hanoi
```

```
inde:delhi
```

- la commande

```
$ grep italie villes.txt
```

affiche les lignes du fichier qui contiennent `italie`

villes.txt

france:paris

vietnam:ho chi minh

italie:roma

france:bordeaux

vietnam:hanoi

inde:delhi

Essayer

```
$ grep 'i' villes.txt
```

```
$ grep ':h' villes.txt
```

grep : ancrage

Les caractères ^ et \$ servent à “ancrer” le motif de recherche

- au début d'une ligne
- ou à la fin de la ligne

Essayer

```
$ grep '^i' villes.txt
```

```
$ grep 'i$' villes.txt
```

La commande cut

La commande cut sélectionne une *colonne* de données.

Essayer

```
$ cut -c 1-3 villes.txt
```

```
$ cut -d: -f1
```

```
$ grep vietnam villes.txt | cut -d: -f2
```

la commande sort

Ordonne les lignes selon un critère

```
$ sort villes.txt
```

```
$ sort -t: -k2 villes.txt
```

Exercices “sort”

Exercice

- Créer un fichier `villes-pays.txt` (villes ordonnées par pays)
- Créer un fichier `continents-pays.txt` (continents ordonnées par pays) à partir de `continents.txt`

europa:france

europa:italie

asie:vietnam

asie:chine

Commande “join”

Rapproche deux fichiers sur une **clé commune**.

Les fichiers doivent être triés

Exemple

```
$ join -t: -1 2 -2 1 continents-pays.txt villes-pays.txt
```

- `-t:` : délimiteur de champs
- `-1 2` : clé du premier fichier = second champ
- `-2 1` : clé du second fichier = premier champ

Deuxième partie II

Shell-scripts : introduction

Qu'est-ce qu'un shell ?

Utilisation interactive / scripts

Comment écrire un script

Démonstration

Pourquoi écrire des scripts ?

Qu'est-ce qu'un shell ?

Qu'est ce qu'un *shell*

«shell» : programme qui

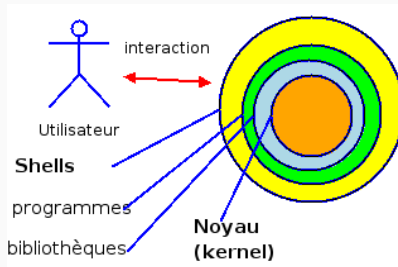
- lit des lignes de commandes
 - tapées par l'utilisateur
 - ou lues depuis un fichier
- les fait exécuter

Autre nom : interprète de commandes

Les shells (suite)

Les shells

- ne font pas partie du noyau du système,
- utilisent le noyau pour exécuter des applications, créer des fichiers etc.



Les shells

De nombreux shells sont disponibles sous Unix/Linux,

- sh
- **bash** (Bourne again shell),
- CSH (C Shell),
- KSH (KORN Shell),
- TCSH
- ...

Ils

- jouent le même rôle,
- ont des **syntaxes** différentes,
- fournissent des **fonctions prédéfinies** différentes.

En pratique

- Le fichier `/etc/shells` contient la liste des shells disponibles
`$ cat /etc/shells`
- pour savoir quel shell vous utilisez, tapez
`$ echo $SHELL`
- Pour connaître votre **shell par défaut**
`$ finger -l`

Utilisation interactive / scripts

Usage interactif

1. vous tapez une commande
2. le *shell* l'interprète
3. vous retournez à l'étape 1

Scripts

1. vous tapez des commandes dans un fichier texte (**script**)
2. vous demandez l'exécution de ce script

Shell script

Définition : Shell-script = fichier texte qui contient une suite de commandes.

Exemple : fichier “premier.sh”

```
1  #!/bin/bash
2  #
3  # Mon premier essai
4  clear
5  echo -n "Nous sommes le "
6  date
7  echo "et c'est mon premier script"
```

Note : le caractère # indique un commentaire

Shell script

Exemple : fichier “premier.sh”

```
1  #!/bin/bash
2  #
3  # Mon premier essai
4  clear
5  echo -n "Nous sommes le "
6  date
7  echo "et c'est mon premier script"
```

Exécution par

```
$ bash premier.sh
```

ou

```
$ chmod +x premier.sh
```

```
$ ./premier.sh
```

Comment écrire un script

Comment écrire un script

1. utiliser un éditeur de textes pour écrire le script

Remarque

- le suffixe `.sh` est recommandé

2. le rendre exécutable

- `chmod +x mon-fichier.sh`
- `chmod 755 mon-fichier.sh`

Démonstration

Démonstration

1. on tape le script suivant

```
1  #!/bin/bash
2  #
3  # fichier premier.sh
4  #
5  clear
6  echo "les scripts, c'est facile"
```

2. exécution par

```
$ bash ./first.sh
```

Recommandation : la première ligne commençant par `#!` indique quel *interpréteur* il faut utiliser.

1. lancement par

```
$ ./first.sh
```

ne marche pas, parce que le script n'est pas exécutable

2. Rendre le fichier exécutable

```
$ chmod +x first
```

```
$ ./first
```

Pourquoi écrire des scripts ?

Pourquoi écrire des scripts ?

Intérêt

- Permettent de réutiliser des suites de commandes sans risque d'erreur
- Gagner du temps
- Automatiser les tâches fréquentes
- ...

Applications

Applications des scripts

- Créer ses propres commandes à partir de commandes existantes
- Scripts d'installation
- Fonctionnement du système d'exploitation (ex : lancement de script au démarrage)
- Aide à l'administration du système (tâche répétitives)
exemple :surveillance des quotas
- ...

Troisième partie III

Variables, paramètres, expressions...

Variables

Environnement, export

Tableaux

Paramètres positionnels

Plus d'informations sur l'affectation

Lecture de variables

Essais

read, exercice

read, IFS

Expansions

Exercices

Une petite application

Affectation du résultat d'une commande

Chaînes et expansion

Variables

Variables

Les variables du *shell* mémorisent des chaînes de caractères.

- la liste des variables est affichée par `set`
- Variables système définies automatiquement (et à connaître) :
HOME PWD SHELL USERNAME PATH LANG
etc.

Affectation / Expansion

- affectation de variable : `NOM=CHAINE`
 - Ne surtout pas mettre d'espace avant et après =
- Accès au contenu par

`$NOM` # *peut être ambiguë*

`${NOM}` # *moins ambiguë*

Exemple

```
message="Bienvenue parmi nous"
```

```
echo $message
```

Variables : exemple

```
1  #!/bin/bash
2
3  echo "Bonjour $USER"
4  echo -n "aujourd'hui "
5  date
6
7  france="Europe/Paris"
8  vietnam="Asia/Ho_Chi_Minh"
9
10 echo -n "heure Vietnam = "
11 TZ=$vietnam date
12
13 echo -n "heure France = "
14 TZ=$france date
```


Variables système

Essayez

Avec la commande `set` et `grep`, trouvez les variables qui indiquent

- le nom de votre poste de travail,
- son type,
- la version du système d'exploitation ?

Environnement, export

Environnement, export

- un processus s'exécute dans un environnement (valeurs de variables). La commande **printenv** affiche les variables qui sont transmises aux processus.
- les variables système sont transmises (**exportées**) automatiquement lors d'un appel de script
- les autres doivent être exportées explicitement (= ajoutées à l'environnement) :

```
export NOM[=VALEUR]
```

Quelques variables utiles

- EDITOR : éditeur de texte par défaut
- PAGER : pageur par défaut
- PATH : liste des chemins de recherche des exécutable
- PS1 : définition du prompt
- VISUAL : idem que EDITOR / avez un sens il y a 30 ans

Tableaux

Tableaux

À la différence de `sh`, l'interprète `bash` possède des `tableaux`

Exemple

```
declare -a pays  
pays[0]=Australie  
pays[1]=Vietnam  
pays[2]=France  
pays[3]=Allemagne
```

```
i=3
```

```
j=1
```

```
echo football :  ${pays[$i]} contre ${pays[$j]}
```

Paramètres positionnels

Paramètres positionnels

Invocation d'un script

Un script peut être **invocé avec des paramètres**, exemple :

```
./mon-script Hanoi Paris Bordeaux "Ho Chi Minh City"
```

Pendant l'exécution

- \$1="Hanoi",
- \$2="Paris",
- \$3="Bordeaux",
- \$4="Ho Chi Minh City"

Paramètres positionnels (suite)

Autres variables utiles

- `$#` = 4 – le nombre de paramètres
- `$*` = Hanoi Paris Bordeaux Ho Chi Minh City
- `$@` = Hanoi Paris Bordeaux "Ho Chi Minh City"
- `$0` = `"/mon-script"` – le nom du script

Exercice

Ecrire un script à un paramètre qui indique les villes d'un pays.

```
$ villes.sh france  
paris  
bordeaux  
...  
$
```

Utiliser le fichier de données précédent.

**Plus d'informations sur
l'affectation**

Affectation

Met une valeur dans une variable.

Noms de variable

- lettres, des chiffres, blancs soulignés
- ne commence pas par un chiffre
- MAJUSCULES/minuscules différenciées

Affectation (suite)

Deux formes

- `NOM=CHaine`
affectation simple
- `let NOM=EXPRESSION`
affectation du résultat d'un calcul arithmétique

`let` est une commande interne du *bash* (`help let`)

Affectations (suite)

Essayez

```
a=12
```

```
b=42
```

```
c=a+b
```

```
echo $c
```

```
d=$a+$b
```

```
echo $d
```

```
let e=a+b
```

```
echo $e
```

Lecture de variables

Lecture de variables

La commande

```
read v1 v2 ...
```

lit

une ligne au terminal, et affecte les mots dans les variables citées.

Exemple

```
read nom prenom
```


Essayez

- `read nom prenom`
- que se passe-t-il si on tape plus de mots qu'il n'y a de variables ?

Exercice read

Exercice

Écrire un script qui

- demande l'année de naissance,
- affiche l'âge (l'année courante est inscrite en dur dans le script).

Exemple d'exécution

```
$ ./quel_age
```

```
Votre année de naissance ?
```

```
1984
```

```
Vous êtes né en 1984, vous avez donc 33 ans.
```

Séparateur

La variable IFS

indique le séparateur reconnu par read (**input field separator**)

```
$ export IFS=,  
$ read NOM PRENOM  
einstein,albert  
$ echo $PRENOM  
albert
```

Alternative à “export”

Affectation temporaire :

```
IFS=, read NOM PRENOM
```

valide pendant la durée d'exécution du read

Expansions

Expansion

Définition

Expansion : remplacement d'une expression par sa **valeur**

Exemples

- expansion de **variables** :
`echo bonjour $USER`
- expansion **numérique** :
`echo périmetre = $((2*(longueur+hauteur)))`
- expansion du **résultat d'une commande** :
`echo il y a $(who | wc -l) connexions`

Expansion

Autre notation

Historiquement, **sh** utilisait des “anti-quotes” pour `$(...)` :

```
echo il y a `who | wc -l` connexions
```

Moins lisible, risque de confusion avec les apostrophes

```
echo il y a 'who | wc -l' connexions
```

1. Exercice simple

Dans le script qui calcule l'âge, remplacez la **constante 2018** par un appel à date +%Y

2. Mieux

Écrire un script qui affiche le nombre de processus qui vous appartiennent.

Exécution :

```
Sur tuba, adupont a 45 processus
```

Indication : comptez les lignes qui commencent par votre nom dans le résultat de “ps axu”.

3. Encore plus fort

Script qui affiche le nom en clair d'une personne (dont on connaît l'identifiant). Utiliser la commande

```
ldapsearch -x cn=adupont displayname
```

Exercices (suite)

Écrire un script qui indique les 5 plus gros sous-répertoires d'un répertoire donné.

Exécution

```
$ plus-gros.sh ~/Essais
196      /home/billaud/Essais/LATEX
152      /home/billaud/Essais/C++
96       /home/billaud/Essais/Python
36       /home/billaud/Essais/PHP
```

Indications

- script à 1 paramètre
- `du -s repertoire/*`
- tri *numérique*
- commande `tail -n nombre`

Une petite application

Application : carnet de téléphone

Sous forme de trois commandes

- `tel-ajouter numero nom`
- `tel-chercher nom`
- `tel-afficher`

qui agissent sur un fichier de données `telephones.dat`

Format : un numéro et un nom par ligne

exemple

01234578 PUF

98765444 Charlie

application (suite)

```
#!/bin/bash
# tel-afficher
#
nomFichier="telephones.dat"
cat $nomFichier
```

```
#!/bin/bash
# tel-ajouter numero nom
#
nomFichier="telephone.dat"
echo $@ >> $nomFichier
```

```
#!/bin/bash
# tel-chercher nom
#
nomFichier="telephone.dat"
grep $1 $nomFichier
```

Améliorez la présentation avec la commande dialog

- `dialog --infobox message hauteur largeur`
- `dialog --textbox nomfichier hauteur largeur`
- `dialog --inputbox message hauteur largeur`

Attention : Avec une “inputbox”, le résultat va sur la sortie d'erreur.

Affectation du résultat d'une commande

Commande et variable

- Ne pas confondre

<code>v=date</code>	affectation de la chaine "date"
<code>date > f</code>	redirection de la sortie vers un fichier
<code>v=\$(date)</code>	affectation de la sortie dans une variable

- Exercice : que fait ceci

```
$cmd > $f
```

?

Exercice : que fait ceci

```
$cmd > $f
```

?

Exemple

```
#
```

```
format="%Y-%M-%d"
```

```
cmd="date +%format"
```

```
f=/tmp/resultat
```

```
$cmd > $f
```

Chaînes et expansion

Chaînes et expansion

L'expansion

- se fait dans les chaînes délimitées par `"..."`
- pas dans les chaînes délimitées par `'...'`

Exemple

```
echo 'la variable $USER ' "contient $USER"
```

Quatrième partie IV

Les fonctions

Fonctions

Un script peut comporter des **fonctions**, avec des **paramètres positionnels**

Syntaxe

```
function nom-de-fonction  
{  
    commande  
    commande  
    ...  
}
```

Fonctions : exemple

Exemple

```
#!/bin/bash
```

```
function archiver
```

```
{
```

```
    tar -czf /var/svgd/$1.tgz $2
```

```
}
```

```
archiver photos /home/billaud/photos
```

```
archiver musique /home/billaud/musique
```

Fonctions : avantages

Avantages :

- découpage logique,
- code plus facile à lire
- fonctions réutilisables

Fonctions : exemple

Par défaut, les variables sont communes (globales)

Exemple

```
#!/bin/bash
```

```
destination=/var/svgd
```

```
function archiver
```

```
{
```

```
    tar -czf $destination/$1.tgz $2
```

```
}
```

```
archiver photos /home/billaud/photos
```

```
archiver musique /home/billaud/musique
```

Variables locales

On peut déclarer des **variables locales** dans une fonction

Exemple

```
#!/bin/bash
destination=/var/svgd

function archiver
{
    local nom=$(basename $1)
    tar czf $destination/$nom.tgz
}

archiver /home/billaud/photos
archiver /home/billaud/musique
```

Cinquième partie V

Arithmétique

Let : affectation arithmétique

Exercices

Expansion arithmétique

Let : affectation arithmétique

Let : affectation arithmétique

Syntaxe

```
let VARIABLE=EXPRESSION
```

Exemple à essayer

```
#!/bin/bash
```

```
let somme=$1+$2
```

```
echo $somme
```

Comparer

- `let somme=$1+$2`
- `somme=$1+$2`

Note

Dans une affectation arithmétique, l'expansion des variables est automatique

```
let c=a+b
```

```
let c=$a+$b
```

Exercices

Exercice 1

Ecrire un script qui

- demande l'année de naissance
- affiche l'âge

scenario

```
$ exercice1.sh
```

```
Vous êtes né en quelle année ?
```

```
1990
```

```
Vous avez donc 20 ans
```

```
$
```

Exercice 2

Convertir une heure (donnée sous la forme HHMM) en nombre de minutes

scenario

```
$ exercice2.sh 1015
```

```
615
```

```
$
```

Exercice

Ecrire un script qui calcule la durée d'un trajet, à partir des heures de départ et d'arrivée sous la forme HHMM

Scénario

```
$ ./duree.sh 630 2215  
1545
```

Expansion arithmétique

Expansion arithmétique

A la place de

```
let surface=hauteur*largeur  
echo la surface du rectangle est $surface m2
```

On peut écrire

```
let surface=hauteur*largeur  
echo la surface du rectangle est $((largeur*hauteur)) m2
```

Sixième partie VI

Structure de contrôle : case

Présentation

Exemple

Motifs d'un case

Présentation

Structure de contrôle “case”

Choisit les commandes à exécuter en fonction d'un sélecteur

- semblable au “switch” de C++
- le sélecteur est une chaîne de caractères

Exemple sérieux

```
#!/bin/bash
```

```
# Usage : archiver nom-de-répertoire
```

```
echo "Format = normal gz ?"
```

```
read format
```

```
case "$format" in
```

```
gz)
```

```
    option=z ;   suffixe=tgz   ;;
```

```
normal | "" )
```

```
    option= ;    suffixe=tar   ;;
```

```
*)
```

```
    echo "format '$format' non reconnu" >&2
```

```
    exit 1
```

```
;;
```

```
esac
```

```
prefixe=$(basename $1)
```

```
tar -c${option}f $prefixe.$suffixe $1
```

Exemple

case

Exemple

```
#                               ;;
# usage :                       chercher)
# tel ajouter num nom          grep "$2" $nomFichier
# tel chercher nom             ;;
# tel voir                     voir)
#                               cat $nomFichier
nomFichier="telephone.dat"      ;;
                                *)
case "$1" in                   echo "Erreur"
ajouter)                       exit 1
    shift
    echo $* >> $nomFichier     esac
```

Motifs d'un case

Motifs d'un case

Plusieurs motifs pour un même cas

```
case $reponse in
  oui | o )
    echo "d'accord"
    ;;
  non | n )
    echo "tant pis"
    ;;
esac
```

Motifs d'un case jokers

Utilisation des “jokers” de bash

```
case $reponse in
[oO][Uu][iI] | [oO] )
    echo "d'accord"
    ;;
[nN][oO] )
    echo "tant pis"
    ;;
*)
    echo "quoi ?"
    ;;
esac
```

Septième partie VII

Processus

Définitions

Table des processus

kill

Pilotage des processus

Une application

Un exemple de service

Le code principal

Les fonctions

Définitions

Définitions

Un processus = un programme “qui tourne”

Un programme lancé depuis le shell peut

- tourner en “avant plan” (foreground) : il faut attendre sa fin pour lancer une autre commande
- tourner en “arrière-plan” (background)
- être stoppé

Pour

lancer une commande en avant-plan	xclock
stopper la commande en avant-plan	CTRL-Z
relancer la commande stoppée en avant-plan	fg
relancer la commande stoppée en arrière plan	bg
lancer une commande en arrière-plan	xclock &

Note : Si il y a plusieurs commandes en arrière-plan, commandes

- jobs,
- fg %n,
- etc.

Table des processus

La table des processus

On peut la voir par la commande `ps`, ou `top`, ...

Exemple

```
$ ps
  PID TTY          TIME CMD
 4056 pts/1    00:00:00 bash
 4236 pts/1    00:00:07 xpdf.bin
 4243 pts/1    00:00:10 emacs
 4471 pts/1    00:00:00 xterm
 4613 pts/1    00:00:00 ps
```

- `ps` sans option montre les processus issus du shell
- options intéressantes : `axule...`
- voir aussi `pstree`

kill



La commande “kill”

- Syntaxe : `kill [-signal] num-processus ...`
- Rôle : envoie un **signal** à des processus

Exemple

```
$ xclock -digital -update 1 &  
[6] 4734
```

```
$ ps  
  PID TTY          TIME CMD  
 4056 pts/1    00:00:00 bash  
 4236 pts/1    00:00:07 xpdf.bin  
 4734 pts/1    00:00:00 xclock  
 4739 pts/1    00:00:00 ps
```

```
$ kill -TERM 4734
```


La commande “kill” (suite)

- Par défaut, utilise le signal TERM (9) qui termine le programme.
- le signal STOP arrête un processus
- le signal CONT le relance
- `kill -l` affiche la liste des signaux

Pilotage des processus

Pilotage des processus

- commande `&` lance une commande en arrière-plan
- la variable `$!` contient son numéro de processus
- la variable `$$` = numero du shell courant

Exemple

```
mplayer funny-music.mp3 >/dev/null &  
music=$!
```

```
# sauvegardes
```

```
tar czf .....
```

```
# arrêter la musique à la fin
```

```
kill -9 $music
```

Wait

`wait` nnn attend un processus

Exemple

```
mpplayer funny-music.mp3 >/dev/null &  
music=$!
```

```
# sauvegardes en parallèle
```

```
tar czf archive1.tar ..... &  
svgd1=$!
```

```
tar czf archive2.tar ..... &  
svgd2=$!
```

```
wait $svgd1
```

```
wait $svgd2
```

```
kill -9 $music # arrête la musique
```

Une application

Un exemple de service

Une commande pour faire apparaître / disparaître une pendule sur le bureau

Usage

- `./pendule.sh start`
- `./pendule.sh stop`
- `./pendule.sh usage`
- `./pendule.sh restart`

Le code principal

Code 2/2

```
case "$1" in
start)
    do_start ;;
stop)
    do_stop ;;
restart)
    do_stop
    do_start ;;
usage)
    print_usage ;;
*)
    print_usage
    exit 1
esac
```

Constantes et Fonctions

Code 1/2

```
prog=/usr/bin/xclock  
pid_file=/tmp/$USER.pid
```

```
function do_start {  
    $prog &  
    echo $! > $pid_file  
}
```

```
function do_stop {  
    kill -9 $(cat $pid_file)  
}
```

```
function usage {  
    echo "usage: pendule {start|stop|restart|usage}"  
}
```


Huitième partie VIII

Boucle for

Boucle for

Exercice

Exercice (seq)

Exercice (find)

Boucle for

Boucle for

Forme générale

```
for VAR in LISTE
do
    COMMANDE
    COMMANDE
    . . . .
done
```

Boucle avec une variable qui parcourt une liste de mots.

Boucle for, exemples simples

```
for f in *.cc
do
    astyle --style=gnu $f
done
```

```
for f in *.cc
do
    echo "le fichier $f contient $(wc -l $f) lignes"
done
```

Exercise

Boucle for, exercice

Ecrire une commande qui calcule la somme de ses paramètres (en nombre illimité)

```
$ somme 100 3 20
```

```
123
```

Exercise (seq)

Exercice (seq)

Analyser le script

```
r=1
for i in $(seq 1 $1)
do
    let r*=i
done
```

Exercise (find)

Exercice (find)

Analyser le script

```
for f in $(find ~ -name '*.cc' -ctime -7)
do
    ls -l "$f"
done
```

- rôle de `find ~`
- rôle de `find ~ -name '*.cc'`
- rôle de `find ~ -name '*.cc' -ctime -7r`
- pourquoi les guillemets dans `ls` ?

Neuvième partie IX

Décisions

Code de retour (exit status)

if-then-else-fi

La commande test

Enchaînement conditionnel

if-then-elif-else, forme générale

Code de retour (exit status)

Code de retour (exit status)

- Le **code de retour** d'un programme indique si il s'est bien terminé.
- Le code de retour de la dernière commande est dans la variable `$?`
- Par convention : `0 = OK`.

code de retour

Exemple

```
$ ls -ld /tmp
drwxrwxrwt 10 root root 12288 déc 22 17:32 /tmp
$ echo code de retour = $?
code de retour = 0
```

```
$ ls -l qsdsqd
ls: ne peut accéder qsdsqd: Aucun fichier ou
répertoire de ce type
$ echo code de retour = $?
code de retour = 2
```


Codes de retour et documentation

Les codes de retour des commandes sont décrits dans le manuel

Exemple : `man ls`

...

```
Exit status is 0 if OK, 1 if minor problems,  
2 if serious trouble.
```

- Par défaut, un script retourne le code de sa dernière commande
- Il peut retourner un code spécifique par
`exit [code]`

Exemple

```
#!/bin/bash  
echo "Something strange happened"  
exit 42
```

if-then-else-fi

La structure de contrôle `if then else fi` utilise le code de retour d'une commande

Exemple

```
#!/bin/bash
# usage:
#   compiler.sh prefixe
#
if g++ -o $1 $1.cc
then
    echo "la compilation s'est bien passée"
else
    echo "il y a eu un problème"
fi
```

La commande test

Test

Le code de retour du programme test dépend d'une **condition**.

Exemple :

```
test -f nomFichier
```

indique si le fichier existe.

Application

fichier "compiler.sh"

```
#!/bin/bash
```

```
if test -f $1.cc
```

```
then
```

```
    g++ -Wall -o $1 $1.cc
```

```
    echo Compilation terminée
```

```
else
```

```
    echo "Erreur: pas de fichier $1.cc"
```

```
    exit 1
```

```
fi
```

Autre notation : [condition]

Autres tests :

[-d nom]	# nom désigne un répertoire
[chaine1 = chaine2]	# comparaison de chaines
[chaine1 != chaine2]	
[chaine1 \< chaine2]	
[chaine1 \> chaine2]	
[nombre1 -eq nombre2]	# comparaison de nombres
[nombre1 -ne nombre2]	
[nombre1 -le nombre2]	# less or equal
[nombre2 -ge nombre2]	# greater or equal
	# voir aussi -lt et -gt

Exercice

Ecrire une commande qui affiche le maximum de deux paramètres.

Scénario

```
$ max.sh 37 421
```

```
421
```

Enchaînement conditionnel

Enchaînement conditionnel

Syntaxe

COMMANDE1 && COMMANDE2

COMMANDE1 || COMMANDE2

Exécute la seconde commande seulement si la première a réussi (&&) ou échoué (||)

Exemple

```
g++ prog.cc && echo OK
```

Application

```
#!/bin/bash
```

```
max=$1
```

```
test $2 -ge $max && max=$2
```

```
echo $max
```

if-then-elif-else, forme générale

if-then-elif-else, forme général

Syntaxe

```
if COMMANDES
then
    COMMANDES
[ elif COMMANDES
  then
    COMMANDES
]...
[ else
    COMMANDES
]
fi
```

Dixième partie X

Boucle while

Boucle while

Boucle while-read

break

Boucle while

Boucle while

- Syntaxe

```
while COMMANDE  
do  
    COMMANDES  
done
```

Exemple

Que fait ce script ?

```
#!/bin/bash
```

```
if test -f $1.cc
```

```
then
```

```
    g++ -Wall -o $1 $1.cc
```

```
    echo Compilation terminée
```

```
else
```

```
    echo "Erreur: pas de fichier $1.cc"
```

```
    exit 1
```

```
fi
```

Boucle while-read

Boucle while-read

Pour traiter le contenu d'un fichier, ligne par ligne.

```
while read numero nom
do
    printf "| %12s | %-30s |\n" $numero $nom
done < agenda.txt
```

break

L'instruction break

permet de sortir d'une boucle

```
while true
do
    . . . .

    echo "voulez-vous arrêter ?"
    read reponse
    test $reponse = oui  && break

    . . .
done
```