

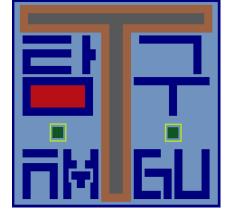
Tamgu (탐구) comme SHELL

Tamgu (탐구) dans son Bac à sable

Claude Roux

Mai 2020

Tamgu 탐구: SHELL

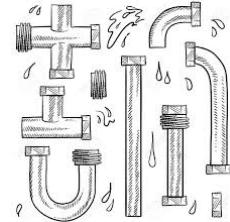


Ô temps! Suspend Ton Vol*



Pipes

Malgré des années d' IUG, il faut quand même avouer que les *pipes* restent assez cool*



```
cat toto | wc -l
```

Hélas! Elles sont parfois *un peu compliquées à manipuler...*

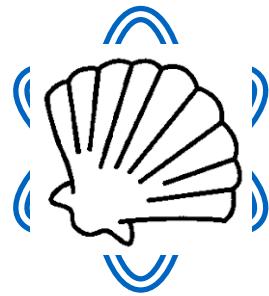
```
-rwxr-xr-x@ 1 roux staff 24373 2 sep 15:55 tamguprimemapsi.cxx
-rwxr-xr-x@ 1 roux staff 24157 2 sep 15:55 tamguprimemapsl.cxx
-rwxr-xr-x@ 1 roux staff 18738 2 sep 15:55 tamguprimemapss.cxx
-rwxr-xr-x@ 1 roux staff 69735 21 oct 14:14 tamgurawstring.cxx
-rwxr-xr-x@ 1 roux staff 27737 2 sep 15:55 tamgusocket.cxx
```

Si nous pouvions juste un instant réfléchir sur le résultat de la dernière commande avant d'agir..



* Je vais vous en mettre pleins les mirettes.

SHELL



Tamgu탐구 est un SHELL*



**Tamgu탐구 est une coquille Saint-Jacques*

Chaines de caractères

```
//Below are some examples on string manipulations
string s;
string x;
vector v;

//Some basic string manipulations
s="12345678a";
x=s[0];           // value=1
x=s[2:3];        // value=3
x=s[2:-2];       //value=34567
x=s[3];          //value=45678a
x=s["56"];       //value=123456
x=s[2:"a"];      //value=2345678a
s[2]='e';        //value=empty

//The 3 last characters
x=s.right(3);    //value=78a

//A split along a space
s='a b c';
v=s.split(" ");   //v=["a","b","c"]

//regex, x is a string, we look for the first match of the regular expression
x=s.scan("%d%d%c"); //value=78a

//We have a pattern, we split our string along that pattern
s='12a23s45e';
v=s.scan(r"%d%d%c");
x=s.replace(r"%d%d%s","X"); //value=12aX45e

//replace also accepts %x variables as in Tamgu regular expressions
x=s.replace(r"%d%1s","%1"); //value=12a2345e

//REGULAR REGULAR EXPRESSIONS: Not available on all platforms
preg rgx(p\w+day);
string str="Yooo Wessdenesday Saturday";
vector vrgx=rgx in str; //["Wessdenesday","Saturday"]

string s=rgx in str; //Wessdenesday
int i=rgx in str; // position is 5

//We use (...) to isolate specific tokens that will be stored in the
//vector
rgx=p(\d{1,3}):(d{1,3}):(d{1,3}):(d{1,3})';
str=1:22:33:444';
vrgx=str.split(rgx); // [1,22,33,444], rgx is a split expression

str=1:22:33:4444';
vrgx=str.split(rgx); // [] (4444 contains 4 digits)

str="A_bcdE";
//Full match required
if (p[a-zA-Z]_+ == Str)
  println("Yooo"); //Yooo
```

D'abord, il manipule les chaines de caractères comme un pro

Il sait tout sur les chaines

- *Encodage (Latin, UTF8, UNICODE)*
- *Expressions régulières*
- *Sous-chaines*
- *Recherche*
- *Segmentation*
- *Découpage*
- *Conversion*

Et il est facile

Vraiment...

Comme Python mais en mieux...

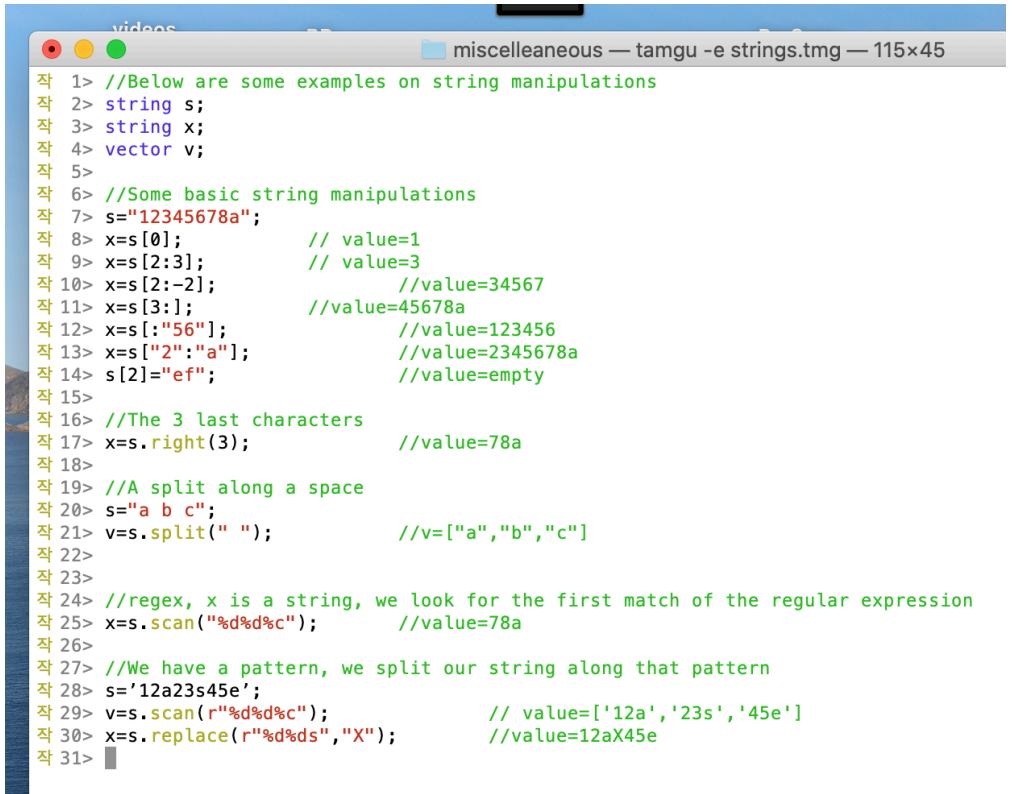


*Tamgu 탐구 est un langage moral indexé sur des shorts

Chaines ?

Ok... Je croyais qu'on allait parler d'un SHELL...

Alors les chaines?



```
videos miscellaneous — tamgu -e strings.tmg — 115x45
작 1> //Below are some examples on string manipulations
작 2> string s;
작 3> string x;
작 4> vector v;
작 5>
작 6> //Some basic string manipulations
작 7> s="12345678a";
작 8> x=s[0];           // value=1
작 9> x=s[2:3];        // value=3
작 10> x=s[2:-2];      //value=34567
작 11> x=s[3:];         //value=45678a
작 12> x=s[:"56"];;     //value=123456
작 13> x=s["2":"a"];     //value=2345678a
작 14> s[2]="ef";       //value=empty
작 15>
작 16> //The 3 last characters
작 17> x=s.right(3);    //value=78a
작 18>
작 19> //A split along a space
작 20> s="a b c";
작 21> v=s.split(" ");    //v=["a","b","c"]
작 22>
작 23>
작 24> //regex, x is a string, we look for the first match of the regular expression
작 25> x=s.scan("%d%d%c"); //value=78a
작 26>
작 27> //We have a pattern, we split our string along that pattern
작 28> s='12a23s45e';
작 29> v=s.scan(r"%d%d%c"); // value=['12a','23s','45e']
작 30> x=s.replace(r"%d%ds","X"); //value=12aX45e
작 31>
```

Pour les *pipes*, TOUT est *chaines*. Chaque processus passe au suivant son résultat sous la forme de chaines.

CHAINES DE CARACTÈRES*



Puis-je revoir cette image encore une fois?

The screenshot shows a Mac OS X window titled "miscelleaneous — tamgu -e strings.tmg — 99x32". The code editor displays the following Python-like pseudocode:

```
작 5>
작 6> //Some basic string manipulations
작 7> s="12345678a";
작 8> x=s[0];           // value=1
작 9> x=s[2:3];         // value=3
작 10> x=s[2:-2];        //value=34567
작 11> x=s[3:];          //value=45678a
작 12> x=s[:"56"];        //value=123456
작 13> x=s["2":"a"];       //value=2345678a
작 14> s[2]="ef";         //value=empty
작 15>
작 16> //The 3 last characters
작 17> x=s.right(3);      //value=78a
작 18>
작 19> //regex, x is a string, we look for the first
작 20> x=s.scan("%d%d%c"); //value=12e
작 21>
작 22> //A split along a space
작 23> s="a b c";
작 24> v=s.split(" ");      //v=["a","b","c"]
작 25>
작 26> //We have a pattern, we split our string along that pattern
작 27> s='12a23s45e';
◆34> exit editor
```

A red circle highlights the command `◆34>`. Below it, the variable `s` is set to `12a23s45e`, and the variable `x` is set to `12aX45e`.

**J'aime bien les couleurs.
C'est quoi cet éditeur?**

**C'est Jag^작 l'éditeur interne de Tamgu
탐구!!!**

Il est *interactif...*

**Vous pouvez exécuter, débogguer, et
visualiser vos variables...**



Jag작: L'éditeur de Tamgu탐구

Jag작* est l'éditeur personnel de Tamgu탐구
Il est aussi accessible directement via: [jag](#)

Commands:

- **3. edit (space):** edit mode. You can optionally select also a file space
 - **Ctrl-b:** toggle breakpoint
 - **Ctrl-k:** delete from cursor up to the end of the line
 - **Ctrl-d:** delete a full line
 - **Ctrl-u:** undo last modification
 - **Ctrl-r:** redo last modification
 - **Ctrl-f:** find a string
 - **Ctrl-n:** find next
 - **Ctrl-g:** move to a specific line, '\$' is the end of the code
 - **Ctrl-l:** toggle between top and bottom of the screen
 - **Ctrl-t:** reindent the code
 - **Ctrl-h:** local help
 - **Ctrl-w:** write file to disk
 - **Ctrl-c:** exit the editor
- **Ctrl-x: Combined Commands**
 - **C:** count a pattern
 - **H:** convert HTML entities to Unicode characters
 - **D:** delete a bloc of lines
 - **c:** copy a bloc of lines
 - **x:** cut a bloc of lines
 - **v:** paste a bloc of lines
 - **d:** run in debug mode
 - **r:** run the code
 - **w:** write and quit
 - **l:** load a file
 - **m:** display meta-characters
 - **h:** full help
 - **q:** quit



Vous pouvez exécuter vos commandes directement dans le SHELL

Tamgu 0.96.4 build 55(탐구)

Copyright 2019–present NAVER Corp.
64 bits

!commande*

help: display a list of available commands

◀▶2> !ls
buggui.tmg
build.xml
built.xml
centos.sh
checkps.tmg
chrono.tmg
cpjag.sh
english.tra
exempleregles.tmg
extraitsderegles.tmg
◀▶2>

fedora.sh
french.tra
gram.tmg
gros.txt
grosvecteurs.tmg
hst.txt
lectecr.tmg
lg.txt
listecode.tmg
remplace_blog.tmg



Encore mieux... Vous pouvez garder le résultat de vos commandes

```
[ ◀2> !v=ls *.tmg
[ ◀2> v
['buggui.tmg
', 'checkps.tmg
', 'chrono.tmg
', 'exempleregles.tmg
', 'extraitsderegles.tmg
', 'gram.tmg
', 'grosvecteurs.tmg
', 'lectecr.tmg
', 'listecode.tmg
', 'remplace_blog.tmg
', 'retirecopyrights.tmg
', 'règles tokenization.tmg
', 'rings.tmg
', 'test.tmg
', 'testconversion.tmg
', 'testlecture.tmg
', 'trialahanoi.tmg
', 'vitesseconversion.tmg
']
[ ◀2>
```

`!v=commande*`

Juste la
commande après
le signe "="...
Pas de quotes...

`v` est un vecteur pré-déclaré.
Mais vous pouvez en déclarer
autant que vous voulez.



* Encore une fois, ça se passe au-dessus

Les commandes peuvent être plutôt compliquées

!v=ps -elf*

```
2> !v=ps -elf
```

```
2> v
```

	UID	PID	PPID	F	CPU	PRI	NI	SZ	RSS	WCHAN
,	0	1	0	4004	0	37	0	5407292	28092	-
,	0	111	1	4004	0	4	0	4482128	1356	-
,	0	112	1	4004	0	31	0	5429284	13088	-
,	0	115	1	4004	0	20	0	4308708	2352	-
,	0	116	1	4004	0	46	0	5935016	28584	-
,	0	117	1	1004004	0	50	0	5405368	14228	-
,	0	118	1	4004	0	4	0	4512256	13300	-



Historique

```
❷ 2> history
```

```
1 = !v=ls *.tmg
2 = v=_sys.pipe("ls *.tmg");
3 = v
4 = !v=ps -elf
5 = v=_sys.pipe("ps -elf");
6 = v
7 = history
```

!v=ls *tmg

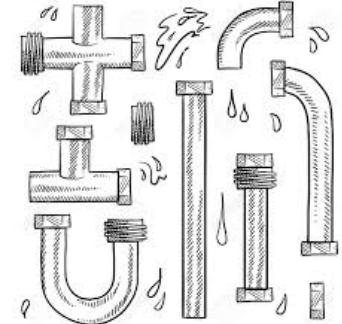
!v=ps -elf

```
❷ 2> store myhistory.hst
```

- Vous avez accès à votre *historique de commandes** (via les flèches)
- Vous pouvez le sauvegarder...
- Encore mieux. Vous pouvez le recharger...



Pipes* dans *Tamgu*



Il existe plusieurs façons d'exécuter vos commandes

Commençons par la plus simple: "-p"

```
ls -1 | tamgu -p "if ('.tmg' in l) println(l);"
```

Et là.. "*l*" ? D'où vient-il?

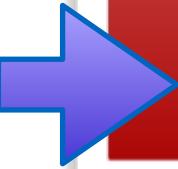


-p

Lorsque vous utilisez "-p", les variables suivantes sont automatiquement pré-déclarées:

_args: argument vector
_paths: _paths[0] is the current directory
a,b,c: bool
i,j,k: int
f,g,h: float
s,t,u: string
m: map
v: vector
x,y,z: self
l: string (current line from stdin for -p)

l: string (ligne courante dans stdin)



Ce qui est bien pratique...



* En allemand, "handy" c'est un téléphone portable. Comme quoi on n'est pas les seuls à avoir des problèmes avec l'anglais...

-pb, -pe: Début et Fin

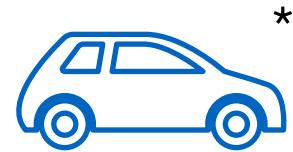
Nous allons compter les fichiers qui ont l'extension: .tmg

```
ls -1 | tamgu -pb "i=0;" -p "if ('.tmg' in l) i++;" -pe "println(i);"
```

- Avec **-pb** vous initialisez "i" avec 0
- Avec **-p** vous captez chaque ligne depuis *stdin*
- Avec **-pe** vous affichez le résultat final



Sous le capot



Nous avons les variables pré-déclarées suivantes:

- `bool a,b,c;`
- `int i,j,k;`
- `float f,g,h;`
- `string s,t,u;`
- `map m;`
- `vector v;`

Ce sont des variables *globales*, dont la valeur se maintiendra quelque soit le nombre d'itération.

```
string l;
```

"l" est une variable locale...

En fait, chaque fois qu'une nouvelle ligne est détectée dans `stdin`, "-p" appelle une fonction dont "l" est l'argument.



Sous le capot



En réalité, c'est encore plus intéressant...

Chaque ligne provenant de *stdin* est aussi découpée selon les espaces qu'elle contient et des variables l1..l99 sont alors automatiquement initialisées avec chaque champ.

De plus, `_args` est un vecteur de chaînes avec l'équivalence suivante:

```
_args[0] <=> l  
_args[1] <=> l1  
_args[2] <=> l2  
_args[3] <=> l3
```

etc...

La variable "`_size`" contient le nombre total de champs extraits...



* Je sais, j'ai honteusement piraté awk...

GREP

Vous pouvez implanter votre propre *grep**.

"s" est une chaîne de caractères, qui expose la méthode:
read, laquelle permet de lire le contenu d'un fichier
directement dans une chaîne.

```
ls -1 | tamgu -p "if ('.txt' in l) {s.read(l); if ('TOTO' in s) println(l);}"
```

Affiche le nom des fichiers .txt qui contiennent *TOTO*



* La vaccination contre la grep est conseillée pour les séniors.

-a: Gardez vos arguments près de vous...

-a enregistre le contenu de stdin dans: _args*.

```
ls -1 | tamgu -a
```

Tamgu 0.96.4 build 55(탐구)

Copyright 2019–present NAVER Corp.

64 bits

help: display available commands

[◀▶2> _args

['french.tra' , 'generation.tmg' , 'generationbis.tmg']

◀▶3>



Jouons un peu...*

```
►2> _args
['french.tra','generation.tmg','generationbis.tmg']
[]2> for (u in _args) {
[]3>     if (".tmg" in u) {
[]4>         println(u);
[]5>     }
[]6> }
[]7>
generation.tmg
generationbis.tmg
►7> █
```



Vous pouvez éditer vos commandes...*

```
작 1> bool a,b,c; int i,j,k; float f,g,h; string s,t,u; map m; vector v; self x,y,z;
작 2> for (u in _args) {
작 3>     if (".tmg" in u) {
작 4>         println(u);
작 5>     }
작 6> }
```

Notez les variables pré-déclarées



Un gros programme

```
작 1> @@
작 2> _args contains some directories, which are traversed
작 3> for each '.h' file found, we open it and modifies its content
작 4> then save it again in the same file
작 5> @@
작 6>
작 7> function traverse(string chemin) {
작 8>     svector pathname = _sys.ls(chemin);
작 9>     string sub;
작 10>
작 11>     string che;
작 12>
작 13>     for (string n in pathname) {
작 14>         if (n[0]=='.')
                continue;
작 15>
작 16>         che = chemin+n;
작 17>         if (_sys.isdir(che))
                traverse(che+"/");
작 18>         else {
작 19>             if ('.h' in n) {
작 20>                 string mytext;
작 21>                 mytext.read(che);
작 22>                 mytext=mytext.trimleft();
작 23>
작 24>                 file sv(che,"w");
작 25>                 sv.write(mytext);
작 26>                 sv.close();
작 27>
작 28>             }
작 29>         }
작 30>     }
작 31> }
작 32> }
작 33>
작 34>
작 35> for (string c in _args)
작 36>     traverse(c);
작 37>
```

_sys vous donne le pouvoir du SHELL:

- ls(chemin)
- isdirectory(chemin)
- command(...)
- pipe(...)

_args n'est jamais détruit*. Vous pouvez exécuter ce programme autant de fois que vous voulez...

