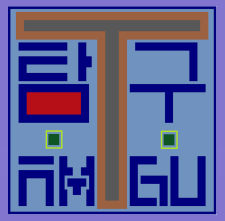


Tamgu (탐구) as a SHELL

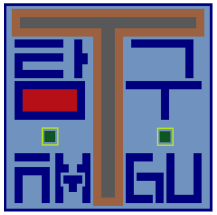
Tamgu (탐구) Playground

Claude Roux

December 2019

LABS

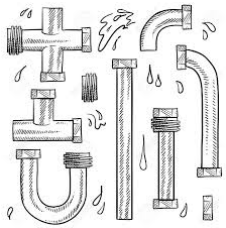
Tamgu 탐구: SHELL



Ô temps! Suspend Ton Vol*...

*Oh Time ! Give us a break !

Pipes



Despite years of GUI, *pipes* are still cool*...

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

Alas! They are sometimes a bit too complex to handle…

```
-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
```

If we could just ponder a little bit on the output of the last command before acting...

** Je vais vous en mettre pleins les mirettes.*

SHELL

Tamgu탐구 is a SHELL*

**Tamgu탐구 est une coquille Saint-Jacques*

STRINGS

```
//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]="ef";              //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");                // value=['12a','23s','45e']
x=s.replace(r"%d%ds","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 Wesdenesday Saturday";
vector vrgx=rgx in str;     //['Wesdenesday','Saturday']

string s=rgx in str;        //Wesdenesday
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
```

First, It is great at handling strings*!!!

It knows everything about strings:

- *Encoding (Latin, UTF8, UNICODE)*
- *Regular expressions*
- *Substrings*
- *Search*
- *Tokenization*
- *Split*
- *Conversion*

And it is easy...

Really...

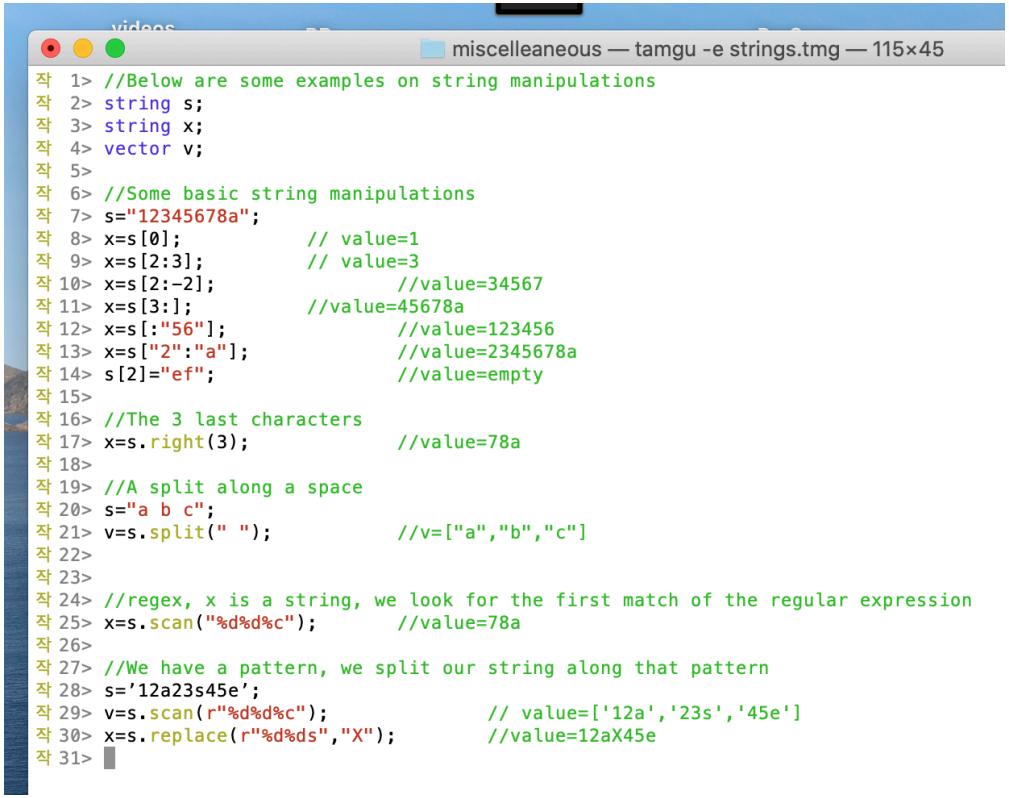
Like Python but better...

*Tamgu탐구 est un langage moral indexé sur des shorts

Strings ?

Ok… But you were talking about a SHELL…

But *Strings* ?



Most of your work, when dealing with *pipes is about strings… This how you pass your output to the next command…*

STRINGS*

* *Encore une fois, Tamgu탐구 est un langage sérieux...*

Can I see the picture again ?

miscelleaneous — tamgu -e strings.tmg — 99×32

```
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

◀▶34>

◀▶33> s
12a23s45e
◀▶35> x
12aX45e
◀▶35>
```

I like the colours, what is this editor ?*

This is Jag작 the internal editor in tamgu탐구!!!

It is *interactive*…

run, debug, introspect your variables...

** Choisir les bonnes couleurs demande un vrai sens artistique, bande de moules...*

Jag작: Tamgu탐구 Editor

Jag작* is the internal editor of Tamgu탐구
It is also available independently from Tamgu탐구 as: jag

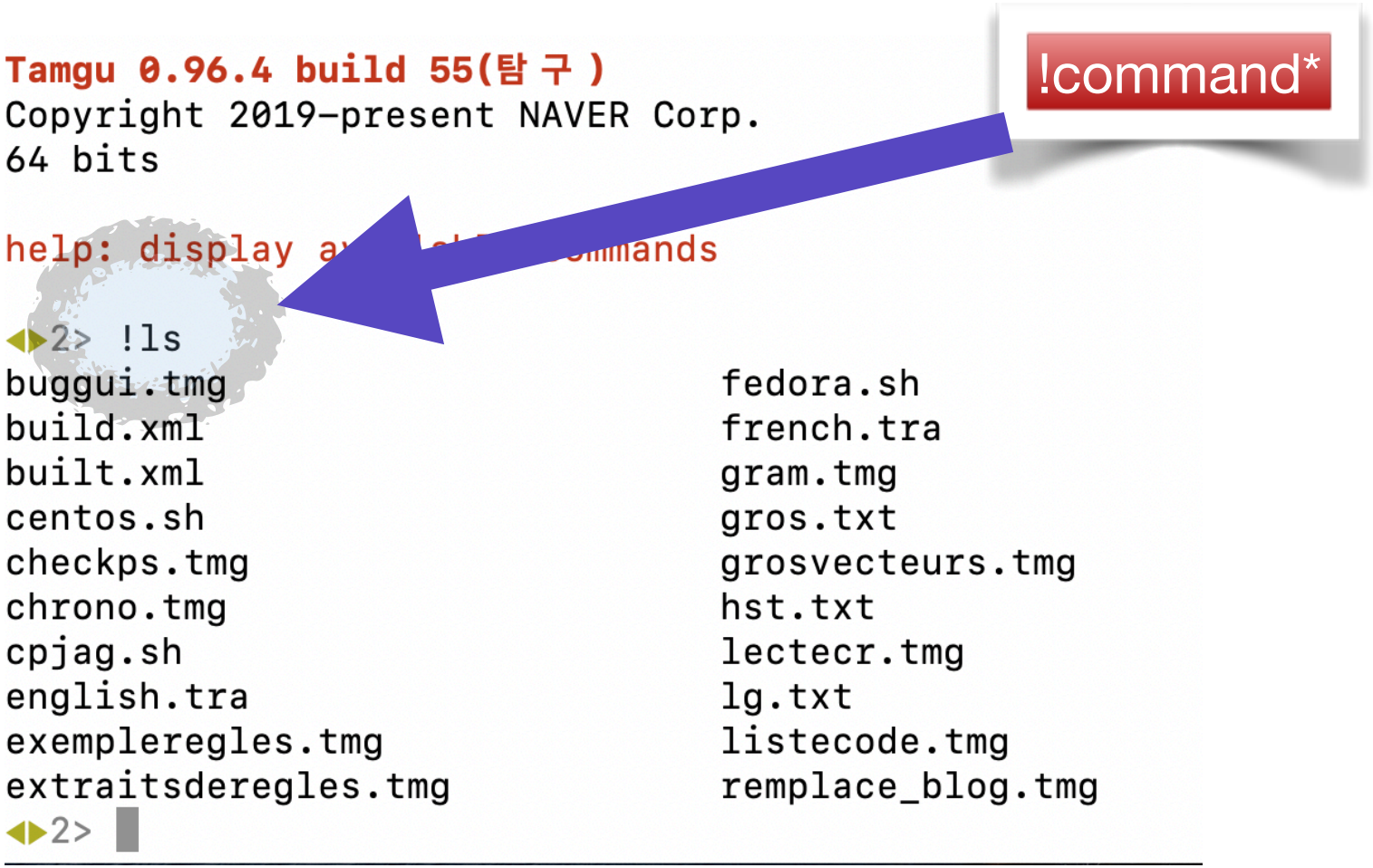
Commands:

- **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

** La type E était vraiment classe. Depuis bof...*

You can execute commands within the SHELL



*Pourriez-vous vous concentrer sur ma présentation SVP.

Even better... You can keep the results of your command

```
[◀▶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>
```

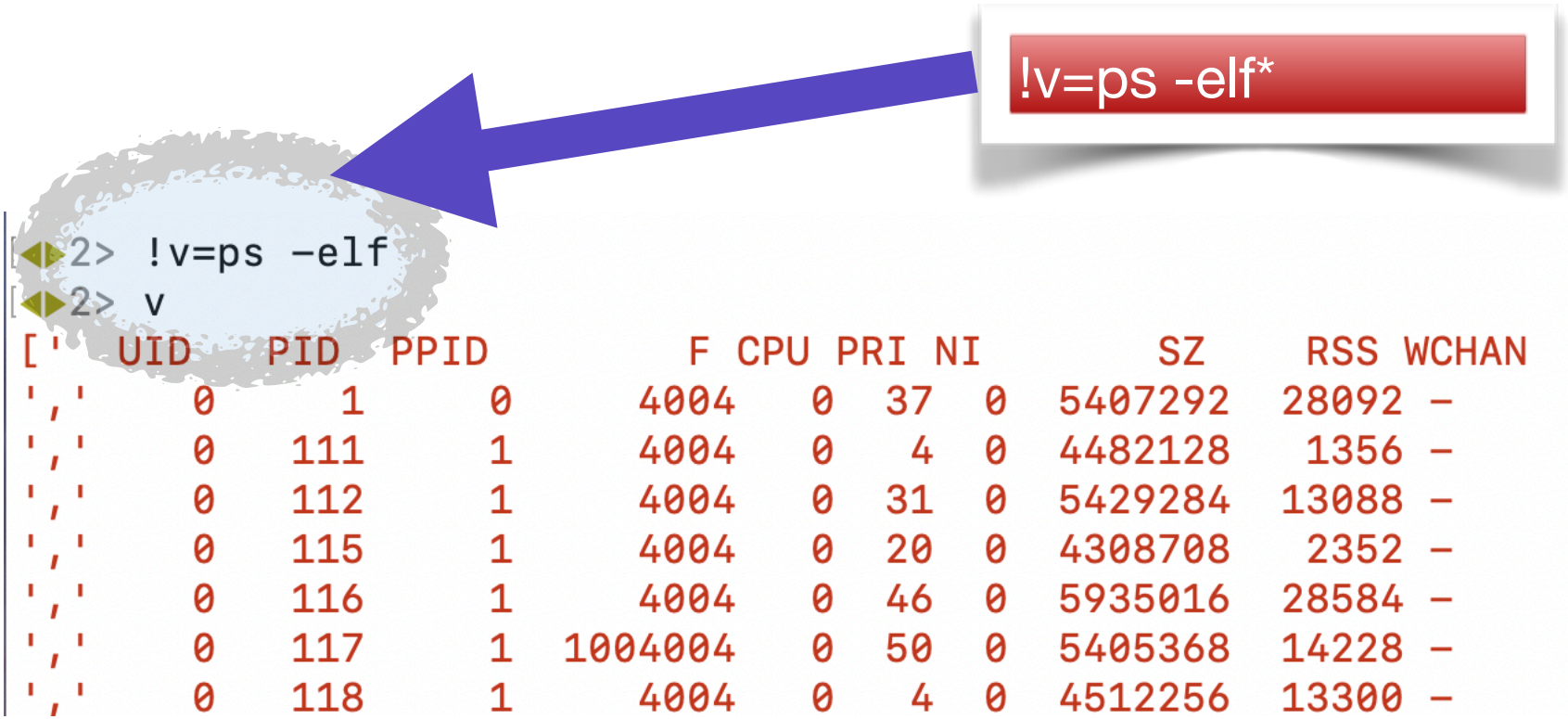
!v=command*

Just the command after the "=" sign... No quote...

v is a predeclared vector... But you can declare as many vectors as you want...

** Encore une fois, ça se passe au-dessus*

These can be quite complicated commands



*Ça dépote hein!!!

History

```
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
2> store myhistory.hst
```

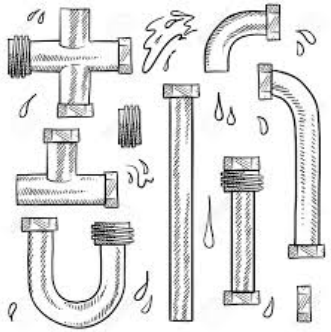
*!v=ls *tmg*

!v=ps -elf

- You can access your *command history** (also with the arrows)
- And you can *store* it...
- Even better. You can re*load* it...

* 1515

Pipes* in *tamgu*



There are many ways to handle your commands.

Let’s start with the simplest: "-p"

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

Where does "l" come from ?

* *Ceci n'est pas une pipe*

-p

Actually, when you use *-p* you have the following variables, which are automatically predeclared for you...

_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 (current line from stdin for -p)

which comes as handy*...

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

-pb, -pe: Beginning and End

Let's say, you want to count the number of files with a specific ending*...

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

- With -pb you initialise "i" with 0
- With -p you scan each line from *stdin*
- With -pe you display the final score

* *Ok... Y'a* "wc -l" *qui fait la même chose. Mais là, c'est plus cool...*

Under the hood

*

We have the following predeclared variables:

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

These are *global* variables, whose values will survive any number of loops.

```
string l;

"l" is a local variable...

Actually, your -p code is encapsulated into a
function, with "l" as a parameter, which is called for
each line from stdin...
```

* *Elle est électrique. C'est important l'économie locale...*

GREP

You can implement your own *grep**.

"s" is a string object, which exposes the method*: read,* which pushes the content of a file into a string variable...

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

Display the *.txt* file pathnames that contain *TOTO*

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

-a: Keep your arguments close to you...

-a stores the output of the previous command into a vector of strings: *_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>
```


* argh...

Now you can play...*

```
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>
```

* *Certainement le transparent le plus nul de la présentation... Désolé...*

You can edit...*

```
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> }
```

Note the predeclared variables

* *Ah pardon! Celui-là est pire... La fatigue je pense...*

You can have a large program

```
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]=='.')
작 15>             continue;
작 16>
작 17>         che = chemin+n;
작 18>         if (_sys.isdirectory(che))
작 19>             traverse(che+"/");
작 20>         else {
작 21>             if ('.h' in n) {
작 22>                 string mytext;
작 23>                 mytext.read(che);
작 24>                 mytext=mytext.trimleft();
작 25>
작 26>                 file sv(che,"w");
작 27>                 sv.write(mytext);
작 28>                 sv.close();
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 gives you the SHELL POWER:

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

_args can NEVER be destroyed*. You can run your program as many times as you want...

* *_args est l'autre nom d'Alain Delon*