Eragon

Geste de savoir

Un shell en écrit en Rust

18 décembre 2020

Table des matières

Voilà depuis quelques temps, si vous me suivez, vous avez peut-être remarqué ou pas(car je le dit peu et que je ne suis pas très actif ici) je m'intéresse beaucoup au langage de programmation Rust, et très récemment j'ai découvert Redox qui est un micro kernel écrit en Rust(je pense faire un billet dessus, si je trouve comment build une iso ou une image pour une vm).
Et en explorant leurs forums et leurs gitlab je suis tombé sur ça ,
un shell écrit en Rust pour Linux et Redox, j'ai été très surpris et en bon utilisateur d'ArchLinux je suis allé voir l'AUR ;
j'y ai trouvé le paquet correspondant `ion`, je l'ai installé et l'ai testé.
EDIT : ion n'est pas mit régulièrement à jour sur l'AUR, installez-le via cargo

1. Introduction à Ion

Ion est donc un shell Unix non-POSIX(si vous ne savez pas ce que c'est je pense que quelques recherches s'imposent) pour Linux mais aussi pour Redox.
Globalement les fonctions simples d'un shell sont là, `ls`, `cd` et d'autres encore. Mais plus intéressant encore c'est son auto-complétion semblable à `fish` ou à `zsh`(deux autres shells).

i

Je me permet de vous donner un peu de documentation car je ne saurais pas tout expliquer.
Le manuel de Ion(présenté à la manière des Rust book)
Le site de Redox
Les forums de Redox(pour poser des questions)
Qu'es-ce qu'un kernel Un microkernel c'est un kernel, mais en plus petit.(enfin… pas vraiment, encore une fois Wikipédia vous aidera grandement)
Il y en à qui ont besoin de liens pour ce que c'est qu'un shell ?
Shell Unix
Ça suffit les liens maintenant, on passe à la suite.

2. Le shell

Ion par défaut ressemble à ça (j'ai eu la flemme de virer mon fichier de config du coup y'a quand même deux/trois modifications par rapport au shell de base)

Et l'auto-complétion aussi y est !

Grâce à screenfetch vous pouvez admirer mon environnement, le prompt est assez dénudé car je n'ai pas encore compris comment le personnaliser, le documentation n'étant pas finie et le projet étant encore tout nouveau.
Malgré cela j'ai quand même personnalisé mon fichier de configuration, qui par défaut se trouve dans `$HOME/.config/ion/initrc`,
la syntaxe n'est pas très différente de celle de bash ou zsh dans leurs fichiers de configuration respectifs.

Voici la partie utilisé de mon fichier de configuration(dans le reste c'est un tas de commentaires pour moi-même)

```
############################
# Ion global configuration #
############################

# History configuration
let HISTORY_SIZE = "100000"
let HISTFILE_SIZE = "100000"
let HISTORY_IGNORE = ["no_such_command", "duplicates"]


##########
# Prompt #
##########

fn PROMPT
    echo -n
        "${c::0x55,bold}${USER}${c::0x4B}\@${c::0x55,bold}$(hostname)${c::defau
        ${c::reset}"
end

################
# Set misc var #
################

# Exported vars
export VISUAL='vim'
export EDITOR='vim'
export PAGER='less'
export HASTE_SERVER="https://haste.breizh.me"
export GTK2_RC_FILES="$HOME/.gtkrc-2.0"
export QT_QPA_PLATFORMTHEME='qt5ct'
export BROWSER="firefox-nightly"
export TERMINAL="termite"
export OPENSSL_INCLUDE_DIR="/usr/include/openssl-1.0"
export OPENSSL_LIB_DIR="/usr/lib/openssl-1.0"

# Paths
export
    PATH="$HOME/.local/bin:$HOME/.cargo/bin/:/usr/bin/core_perl/:$PATH"
export LD_LIBRARY_PATH="/usr/local/lib"

# Ion internal vars
let right_prompt="date '+%m/%d/%y %H:%M:%S'"


###########
# Aliases #
###########

```

```
47 # Color support
48 alias ls="ls --color=auto"
49 alias dir="dir --color=auto"
50 alias vdir="vdir --color=auto"
51 alias grep="grep --color=auto"
52 alias fgrep="fgrep --color=auto"
53 alias egrep="egrep --color=auto"
54
55 # ls, cd, rm, mv...
56 alias sl="ls --color=auto"
57 alias dc="cd --color=auto"
58 alias l="ls -CF --color=auto"
59 alias la="ls -FA --color=auto"
60 alias lla="ls -lrthFA --color=auto"
61 alias ll="ls -lrthF --color=auto"
62 alias cp="cp --interactive"
63 alias mv="mv --interactive"
64 alias df="df --human-readable"
65 alias du="du --human-readable"
66 alias rm="rm -I --preserve-root"
67
68 # Git
69 alias gst="git st"
70 alias gco="git co"
71 alias gadd="git add"
72 alias gico="git co"
73 alias gist="git st"
74 alias giadd="git add"
75
76 # Django
77 alias runserver="pipenv run python manage.py runserver"
78 alias migrate="pipenv run python manage.py makemigrations; pipenv
       run python manage.py migrate"
79 alias manage="pipenv python manage.py"
80
81 # Flask
82 alias fndebug="export FLASK_DEBUG=0"
83 alias frun="flask run"
84
85 # Python
86 alias pep="pycodestyle --show-pep --count"
87 alias pep8="pycodestyle --show-pep --count"
88
89 # Other
90 alias chx="chmod +x"
91 alias alsamixer="alsamixer -c 0"
92 alias qemu-kvm="qemu-system-x86_64 -enable-kvm"
93
94
95 ########################
```

```
# Sourcing other files #
########################

screenfetch
```

Vous reconnaissez probablement la syntaxe de bash ou de zsh, ce qui est normal car cette syntaxe est très proche, rappelez-vous c'est un shell après-tout.

3. La syntaxe

3.0.1. Mais du coup qu'est-ce qui change entre bash ou zsh et ion ?

La syntaxe et le fait que ion ne soit pas POSIX, mais ça ça fait partie de la syntaxe.
Du coup voici vite fait un résumé de ce que j'ai trouvé dans ion et qui change des autres shells.

Les couleurs

```
# Pour définir une couleur vous devrez utiliser le code suivant
echo ${c::0xCouleurEnHexadécimal}
# ou pour passer à la couleur pas défaut
echo ${c::default}
# ou pour réinitialiser toute les modifications
echo ${c::reset}
# ou encore pour définir une couleur en gras
echo ${c::Couleur,bold}
```

La définition des variables

```
## Pour définir une variable on peut utiliser:
# Pour la définir uniquement dans le shell.
let var = "contenu de la variable"
# ou le classique export pour définir une variable globale.
export var = "contenu de la variable"

# Mais aussi
let a b = "contenu de a" "contenu de b"

# Le mot-clé pour supprimer une variable est 'drop'
drop a b
```

Les variables ont des utilisations spéciales dans ion, regardez

```
let foo = "Hello, World"
echo $foo[..5]
echo $foo[7..]
echo $foo[2..9]

# Donne
Hello
World
llo, Wo
```

Une utilisation sympathique du texte et des ”intervalles”(traduction pas très sure de ”range”)

Mais dans ion il existe aussi des array

```
# On crée un array comme ça
let array = [ one two 'three four' ]

# On peu utiliser des méthodes similaires à celles utilisés sur le
    texte
let array = [ 1 2 3 4 5 6 7 8 9 10 ]
echo @array[0]
echo @array[5..=8]

# On peu aussi copier un array dans un autre
let array_copy = [ @array ]

# Effectuer un join sur les éléments d'un array(sur du texte par
    exemple)
let array = [ hello world ]
let as_string = @array

# Mais on peu aussi les concaténer
let array = [1 2 3]
let array ++= [5 6 7]
let array ::= 0

# Ou les utiliser pour passer un argument à une commande
let args = [-l -a --color]
ls @args

# Finalement vous pouvez détruire un array en utilisant 'drop -a'
drop -a array
```

Mais mieux encore... **Non seulement il y à des arrays mais en plus il y à des maps(aussi connues sous le nom dictionnaires)**

```
# Les maps tels qu'elles sont utilisés dans ion viennent tout droit de Rust et sont eput-être un peu complèxes à comprendre, il y en à différents types

# Création d'un/une? HashMap
let hashmap:hmap[] = [ foo=hello bar=world fizz=I buzz=was bazz=here ]

# Création d'un/une? BTreeMap
let hashmap:hmap[] = [ foo=hello bar=world fizz=I buzz=was bazz=here ]

# C'est un dictionnaire donc... on appelle la clé et on à le contenu de cette clé.
let x = bazz
echo @hashmap[bar] @hashmap[$x]

# Ajouter une nouvelle clé(et son contenu)
let x[bork] = oops

# Lister les clés
echo @keys(hashmap)

# Lister les valeurs
echo @values(hashmap)

# Lister les valeurs et les clés en même temps
echo @hashmap
```

Sympa non ? Ion permet donc de faire plein de choses plus poussés les unes que les autres.

Ion supporte aussi les assignations de variables avec le opérateurs mathématiques, mais pas tous, voici la liste de ceux supportés(et non supporté)

```
[x] Addition (+)
[x] Soustraction (-)
[x] Multiplication (*)
[x] Division (/)
[x] Division entière(sans virgule) (//)
[ ] Modulo(reste de la division) (%)
[x] Powers(en anglais car je sais pas) (**)
```

Et on peu assigner les valeurs aux variables comme il suit

```
let value = 0
let value += 5
let value -= 2
let value *= 3
let value //= 2
let value **= 10
let value /= 2

let a b = 5 5
let a b += 3 2
let a b -= 1 1
echo $a $b
```

Les conditions Pas trop de changements de ce côté là, mais voici un exemple

```
# Je ne présente pas les conditions
let x = 5

if test 1 == 1
  let x = 2 # updates existing x
  let y = 3 # defines y
end

echo $x
echo $y
```

Les fonctions Pas trop de changements non-plus, Utilisation des fonctions dans l'exemple précédent.

```
let x = 5

fn print_vars
  echo $x
  echo $y
end

if test 1 == 1
  let x = 2
  let y = 3
  print_vars
end
```

Je laisserais les curieux regarder le manuel(en anglais) pour regarder comment on peu faire fonctionner ça de manière plus poussé.

Voilà, j'ai fait un ~~très~~ rapide tour d'horizon de ce shell.
J'ai ignoré beaucoup de mots-clés, et d'explications(tel que les alias, la configuration… etc)

Peut-être que j'expliquerais tout ça plus en détails dans un autre article/billet.

Donnez-moi vos commentaires et votre avis sur ion, est-ce un shell qui va être oublié ? Quelles sont les fonctionnalités que vous appréciez ?

Personnellement je suit de très près l'avancement de ce shell, car il me plaît et offre un nouvelle alternative aux shells "standards" sur les OS Unix-like.