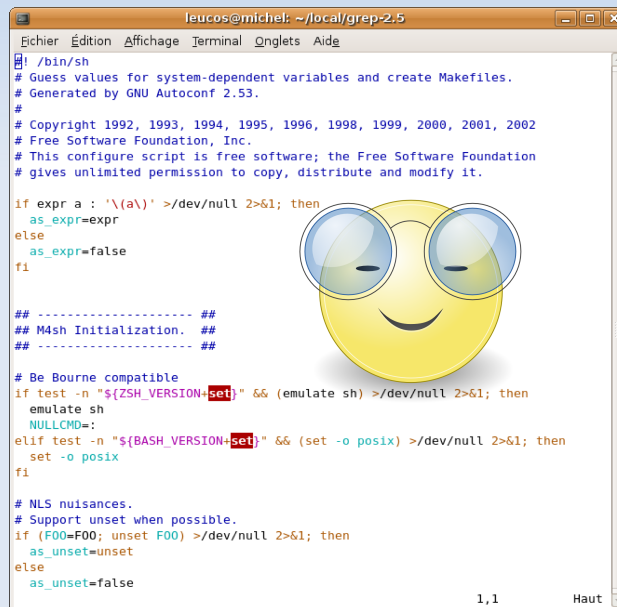


Shell scripting



```
leucos@michel: ~/local/grep-2.5
Fichier  Edition  Affichage  Terminal  Onglets  Aide
#!/bin/sh
# Guess values for system-dependent variables and create Makefiles.
# Generated by GNU Autoconf 2.53.
#
# Copyright 1992, 1993, 1994, 1995, 1996, 1998, 1999, 2000, 2001, 2002
# Free Software Foundation, Inc.
# This configure script is free software; the Free Software Foundation
# gives unlimited permission to copy, distribute and modify it.

if expr a : '\(a\)' >/dev/null 2>&1; then
  as_expr=expr
else
  as_expr=false
fi

## ----- ##
## M4sh Initialization. ##
## ----- ##

# Be Bourne compatible
if test -n "${ZSH_VERSION+set}" && (emulate sh) >/dev/null 2>&1; then
  emulate sh
  NULLCMD=:
elif test -n "${BASH_VERSION+set}" && (set -o posix) >/dev/null 2>&1; then
  set -o posix
fi

# NLS nuisances.
# Support unset when possible.
if (F00=F00; unset F00) >/dev/null 2>&1; then
  as_unset=unset
else
  as_unset=false
fi
```

« *Talk is cheap.
Show me the code* »

Linus Torvalds

Shell scripting

Bases



- Un shell script est un fichier exécutable contenant une série de commandes shell
- Un script commence généralement par :
`#!/bin/sh`
- Il doit être exécutable, ou appelé par 'sh script'
- Par convention, il se termine en général par « .sh »
- Les lignes commençant par '#' sont des commentaires

Shell scripting

Variables



- Dans un script, on peut utiliser des variables locales ou issues de l'environnement

```
echo $HOME
```

- En dehors de l'affectation, la variable doit être précédée par '\$'

```
mavar=bonjour  
echo $mavar
```

- Les arguments passés aussi scripts sont automatiquement affectés aux variables \$1, \$2, ...

Shell scripting

Variables



- La variable `$?` donne la valeur de sortie numérique de la dernière commande
- Un résultat nul indique (en général !) que la commande a réussi

```
user@host:~$ touch f
user@host:~$ cat f
user@host:~$ echo $?
0
```

```
user@host:~$ rm f
user@host:~$ cat f
cat: f: Aucun fichier ou répertoire de ce type
user@host:~$ echo $?
1
```

Shell scripting

Branchements



- La construction if/then/else/fi permet d'effectuer des branchements conditionnels
 - en fonction de valeur de sortie de commandes :

```
echo -n 192.168.0.254 est
if ping -c1 192.168.0.254
then
    echo joignable
else
    echo injoignable
fi
```

Shell scripting

Branchements



- La construction if/then/else/fi permet d'effectuer des branchements conditionnels
 - en fonction de valeur de variables :

```
if [ "$USER" = "root" ]  
then  
    echo Bonjour maitre  
else  
    echo Encore un utilisateur de base...  
fi
```

- La construction [] est l'équivalent de la commande 'test'

Shell scripting

Comparaisons



- La commande `test` (ou `[]`) permet de comparer des valeurs entre-elles
 - Comparaison numériques
 - `-eq` : égal (equal)
 - `-ne` : différent (not equal)
 - `-gt` : plus grand que (greater than)
 - `-ge` : plus grand que ou égal à (greater or equal)
 - `-lt` : plus petit que (less than)
 - `-le` : plus petit que ou égal à (less of equal)
 - Comparaison de chaînes
 - `=` : égal (chaînes identiques)
 - `!=` : différent
 - `<` : précède (tri alphabétique)
 - `>` : suit (tri alphabétique)

Shell scripting

Arithmétique



- Le shell offre trois possibilités pour formuler des calculs :
 - la commande `expr expression` :
`valeur=`expr 10 * 20``
`valeur=`expr $valeur + 2``
 - la construction `$[expression]`
`valeur=${10 * 20}`
`valeur=${$valeur + 2}`
 - la construction `$((expression))`
`valeur=$((10*20))`
`valeur=$(($valeur+2))`

Shell scripting

Boucles



- while condition/do/done permet de boucler tant que la condition est vraie

```
num=1
while [ $num -le 5 ]; do
    echo $num
    num=$(( $num + 1 ))
done
```

```
while ping -c1 $host > /dev/null 2>&1; do
    echo $host est joignable
done
echo $host est Injoignable
```