Unix et Programmation Shell

Philippe Langevin

département d'informatique
UFR sciences et technique
université du sud Toulon Var

Automne 2013

document

brouillon en révision

- site du cours :
  http://langevin.univ-tln.fr/cours/UPS/upsh.html
- localisation du fichier :
  http://langevin.univ-tln.fr/cours/UPS/doc/tools.pdf

dernières modifications

```
     perm.tex 2016-09-30 09:41:54.766553521 +0200
     file.tex 2016-09-30 09:19:02.810595120 +0200
     bash.tex 2016-09-15 12:09:09.887948313 +0200
     term.tex 2016-09-14 18:50:05.124091515 +0200
     upsh.tex 2015-10-25 18:09:36.027434338 +0100
     proc.tex 2015-10-20 22:09:35.450391618 +0200
      man.tex 2015-09-18 20:13:44.822492893 +0200
    shell.tex 2015-09-10 19:31:04.581529236 +0200
prologue.tex 2015-09-07 09:06:31.773157847 +0200
    tools.tex 2015-07-11 09:04:38.890915266 +0200
     pipe.tex 2014-10-02 19:10:22.426127326 +0200
   direct.tex 2014-10-02 07:49:17.162784238 +0200
  syntaxe.tex 2014-10-01 23:52:29.859357485 +0200
     part.tex 2014-10-01 23:52:29.372363438 +0200
```

1 - outils

outils

La liste des commandes utiles s'allonge d'années en années. Mais la croissance n'est pas exponentielle, et il est encore possible de se mettre à jour ! Dans vos TPs, vous travaillerez les incontournables :

- `make`, `gcc`, `gdb`, `gprof` etc...
- `gnuplot`, `dot`, `tex`, `latex`...
- `flex`, `bison`

utiliserez régulièrement les

- `ps`, `sort`, `wc`
- `cut`, `tr`
- `ssh`, `scp`, `sftp`

quelques démos

Il s'agit de faire quelques démonstrations concernant la bibliothèque `regex` et les commandes `grep` , `sed` , `find` , `awk` et `vi`

quelques démos

Il s'agit de faire quelques démonstrations concernant la bibliothèque `regex` et les commandes `grep` , `sed` , `find` , `awk` et `vi`

A très peu de choses prés, tout est prévu dans ces commandes pour les manipulation des fichiers textes. Il s'agit plus de savoir comment retrouver rapidement une option utile avec son mode d'emploi !

quelques démos

Il s'agit de faire quelques démonstrations concernant la bibliothèque `regex` et les commandes `grep` , `sed` , `find` , `awk` et `vi`

A très peu de choses prés, tout est prévu dans ces commandes pour les manipulation des fichiers textes. Il s'agit plus de savoir comment retrouver rapidement une option utile avec son mode d'emploi !

La plupart du temps, les expressions régulières jouent un rôle clef dans le traitement des fichiers.

rexexp

REGEX(7) Linux Programmer's Manual

NAME

regex – POSIX.2 regular expressions

DESCRIPTION

Regular expressions ("RE"s), as defined in POSIX2, come in 2 forms:modern REs (roughly those of egrep; POSIX.2 calls these "extended" REs) and obsolete REs (roughly those of ed(1); POSIX.2 "basic" REs).

Obsolete REs mostly exist for backward compatible. in some old programs; they will be discussed at the end. POSIX.2 leaves some aspects of RE syntax and semantics open; "(!)" marks decisions on these aspects that may not be fully portable to other POSIX.2 implementations.

rexexp

```
1   regcomp (&r, argv[1], REG_EXTENDED) != 0);
2   src  = fopen (argv[2],  "r");
3   while ( ! feof  (src ) ) {
4       fgets  (line ,  1024, src );
5       s  = 0;
6       while (0 == regexec (&r, &(line[s]),  1,  &pos, 0))
7         {
8           // traitement
9         }
10    }
11  regfree (&r);
```

grep

origine

mars 1973, par Ken Thompson.

La version GNU de `grep` utilise l'algorithme de recherche de motif de Boyer-Moore (1977).

- oxford dictionary
- wiki grep
- gnu grep
- voir aussi : `ngrep`

grep

```
NAME
    grep - print lines matching a pattern

SYNOPSIS
    grep [OPTIONS] PATTERN [FILE...]
    grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
    grep searches the named input FILEs for lines
containing a match to the given PATTERN. By
default, grep prints the matching lines.
```

grep -e vs -E

```
↝ grep -rio --include="*.tex"
            -e '\(emacs\|vim\)' ~/doc

↝ grep -rio --include="*.tex"
            -E '(emacs|vim)' ~/doc
```

grep -e vs -E

```
↝ grep -rio --include="*.tex"
            -e '\(emacs\|vim\)' ~/doc

↝ grep -rio --include="*.tex"
            -E '(emacs|vim)' ~/doc
```

- recherche les fichiers TEX, dans l'arborescence **doc**, qui mentionnent un des éditeurs `vi` ou `emacs`

grep -e vs -E

```
↝ grep -rio --include="*.tex"
            -e '\(emacs\|vim\)' ~/doc

↝ grep -rio --include="*.tex"
            -E '(emacs|vim)' ~/doc
```

- recherche les fichiers TEX, dans l'arborescence **doc**, qui mentionnent un des éditeurs `vi` ou `emacs`

```
/home/pl/UPS/doc/tools.tex:emacs
/home/pl/UPS/doc/tools.tex:vim
/home/pl/UPS/doc/prologue.tex:emacs
/home/pl/UPS/doc/prologue.tex:emacs
```

grep -iownE '[-a-z]+regexp' doc/*.tex

```
Matcher Selection
      -E, --extended-regexp
Matching Control
      -e PATTERN, --regexp=PATTERN
      -i, --ignore-case
      -v, --invert-match
      -w, --word-regexp
      -x, --line-regexp
General Output Control
      -c, --count print a count of matching
      -L, --files-without-match
      -l, --files-with-matches
      -m NUM, --max-count=NUM
      -o, --only-matching
      -q, --quiet, --silent
```

grep -Rli –include=*rc path /etc 2>/dev/null

```
Output Line Prefix Control
      –b, ––byte–offset
      –H, ––with–filename
      –h, ––no–filename
      –n, ––line–number
Context Line Control
      –A NUM, ––after–context=NUM
      –B NUM, ––before–context=NUM
      –C NUM, –NUM, ––context=NUM
File and Directory Selection
      ––exclude=GLOB
      ––exclude–from=FILE
      ––exclude–dir=DIR
      ––include=GLOB
      –R, –r, ––recursive
```

exemple

```
↝ grep -iownE '[-a-z]+regexp' doc/*.tex
doc/tools.tex:126:--extended-regexp
doc/tools.tex:128:--regexp
doc/tools.tex:131:--word-regexp
doc/tools.tex:132:--line-regexp
doc/tools.tex:319:--regexp
```

```
↝ grep -iwlER '[-a-z]+regexp' --include=*.tex ~
/home/pl/web-docs/cours/UPS/doc/tools.tex
```

retour de grep

0 au moins une occurrence
1 aucune occurrence
2 erreur d'accès, erreur de syntaxe

```
1  for file in *
2  do
3      if grep -qE '[0-9]+' $file
4          then
5              echo liste de $file :
6              grep -oE '[0-9]+' | sort | uniq -c
7          else
8              echo $file pas de nombre
9      fi
10 done
```

find

origine

Dick Haight (?)

`find` est une commande puissante pour trouver les fichiers en fonction de critères variés.

- sur -print
- wiki find
- gnu findutils
- voir aussi : `locate`, `xargs`

find

La même chose que précédemment, avec `find`, sans le support récursif de `grep`

```
↝ find ~/doc/tools.tex -name "*.tex"
    -exec grep -Hio  -E '(emacs|vim)' {}\;
```

ou encore

```
↝ find ~/doc/tools.tex -name "*.tex" |
          xargs  grep -Hio  -E '(emacs|vim)'
```

man find | wc -l ... 1615 !

```
NAME
   find - search for files in a directory hierarchy
SYNOPSIS
   find [-H][-L][-P][-Dx][-Ox][path...] [expression]

DESCRIPTION
   GNU find searches the directory tree rooted at
each given file name by evaluating the given
expression from left to right, according to the
rules of precedence (see section OPERATORS),
until the outcome is known...
```

option

-maxdepth -mindepth -mount -regextype

```
$ find / -maxdepth 2 -name "$$" 2>/dev/
    null
/proc/2105
$ find / -mount -maxdepth 2 -name "$$" 2>/dev/
    null
```

```
$ find / -maxdepth 2 -regex ".*/[0-2]{1}" 2>/dev
    /null
$ find / -regextype posix-extended -maxdepth 2 \
          -regex ".*/[0-2]{1}" 2>/dev/null
/proc/1
/proc/2
```

```
$ find ~ -mindepth 6 -name '\.*' -prune \
```

test

`-name, -iname`	nom de base	
`-wholename`	nom complet	
`-regex, -iregex`	nom de base	
`-type`	type	f, d, p, c, l, b
`-user`	propriété	
`-group`	groupe	
`-perm`	permission	
`-links`	nombre de liens	
`-size`	taille	char, word, bloc, k, M, G
`-atime,-amin`	date d'accès	
`-mtime,-mmin`	modification	
`-ctime,-cmin`	création	
`-newer`	comparaison des dates	

inoeud faible

```
$ find / -inum -3 -printf "%i %u %p\n"
2 root /
1 pl /home/pl/.gvfs
1 root /sys
2 root /sys/fs
1 root /proc
1 root /proc/sys/fs/binfmt_misc
2 root /proc/sys/fs/binfmt_misc/status
2 root /boot
1 root /dev/pts
2 root /dev/pts/ptmx
```

A propos de `find` et de -**print**

. . . You know, like why is the -**i** option for `grep` mean ignore case, and the -**f** option for `sort` mean ignore case, and so on. . . Well, the instructor of the course decided to chime in and said something like this :

"*Here's another good example of this problem with UNIX. Take the* `find` *command for example. WHAT idiot would program a command so that you have to say* -**print** *to print the output to the screen. What IDIOT would make a command like this and not have the output go to the screen by default*"

And the instructor went on and on, and vented his spleen...

A propos de `find` et de **-print**

The next morning, one of the ladies in the class raised here hand, the instructor called on her, and she proceeded to say something like this :

"*The reason my father programmed the find command that way, was because he was told to do so in his specifications.*"

A propos de `find` et de **-print**

The next morning, one of the ladies in the class raised here hand, the instructor called on her, and she proceeded to say something like this :

"*The reason my father programmed the find command that way, was because he was told to do so in his specifications.*"

- D'après ce courrier, Dick Haight est à l'origine de `find`.

action

```
↝ find ~ -size +1M -ctime -10 -perm 0664
                -name "*.JPG" -user pl -ok delete
    \;
< delete ... /home/pl/pics/IMGP0968.JPG > ? n
< delete ... /home/pl/pics/IMGP0514.JPG > ? n
< delete ... /home/pl/pics/IMGP0965.JPG > ? n
```

```
-delete
-exec command ;
-execdir command ;
-ls
-ok command ;
-okdir command;
-print
-print0
-printf
```

Editeur de flux : sed

```
NAME
   sed - stream editor for filter-transforming text
SYNOPSIS
   sed [OPTION]... {script} [input-file]...
DESCRIPTION
 sed is a stream editor. A stream editor is
used to perform basic text transformations on an
input stream
OPTION
   -n, --quiet, --silent
   -e script, --expression=script
   -r, --regexp-extended
   -f script-file, --file=script-file
   -i[SUFFIX], --in-place[=SUFFIX]
   -l N, --line-length=N
```

sed

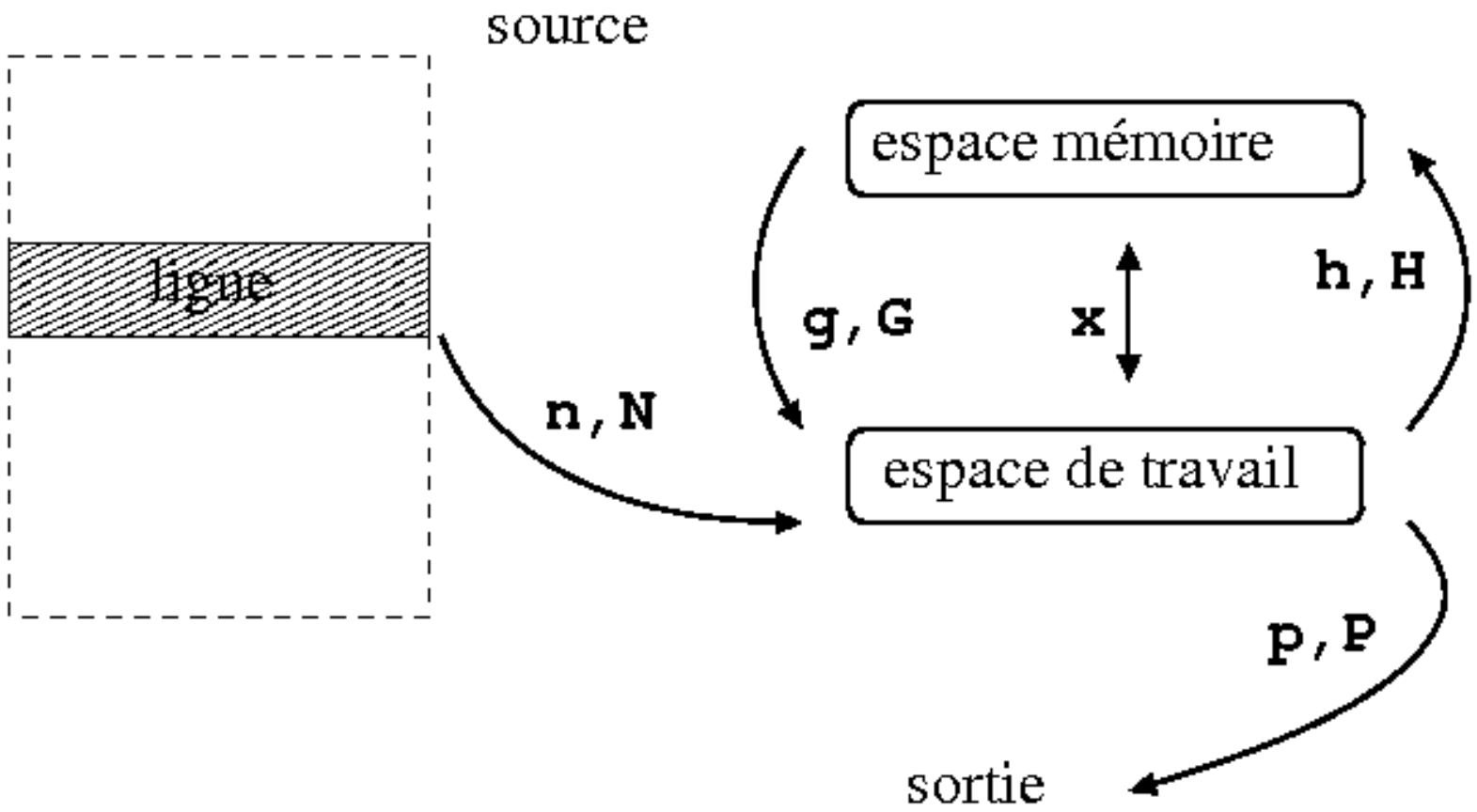
origine

1973-74, par Lee McMahon, Bell labs.

- wiki sed
- gnu sed

principe

`sed` charge les lignes d'un fichier correspondant à une adresse dans un espace de travail pour lui appliquer des commandes d'édition.



adresse : motif, ligne

```
1 #!/bin/bash -v
2 sed '/nologin/d'    /etc/passwd
3 root:x:0:0:  root:/root:/bin/bash
4 sync:x:5:0:  sync:/sbin:/bin/sync
5 shutdown:x:6:0: shutdown:/sbin:/sbin/shutdown
6 halt:x:7:0:  halt:/sbin:/sbin/halt
7 pl:x:501:501::/ home/pl:/bin/bash
8 guest:x:502:502::/ home/guest:/bin/bash
```

adresse : intervalle

```
1  #!/bin/bash -v
2  sed -rn '/dae.*n/,7{=;p}'  /etc/passwd
3  3
4  daemon:x:2:2:daemon:/sbin:/sbin/nologin
5  4
6  adm:x:3:4:adm:/var/adm:/sbin/nologin
7  5
8  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
9  6
10 sync:x:5:0:sync:/sbin:/bin/sync
11 7
12 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
13 24
14 avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-
       daemon:/sbin/nologin
```

commande

q	quitter
d,D	effacer l'espace des motifs
s	substitution
y	remplacement
p, P	affiche l'espace des motifs
n, N	chargement d'une ligne
h, H	mémoriser
g,G	charger la mémoire
x	échange les contenus mémoires
{}	groupement des commandes
=	numéro de ligne
:	étiquette
b	saut

```
1  #!/bin/bash -v
2  echo {1..6} | sed 'y/ /\n/' > /tmp/foo
3  sed 'N;s/\n/-/' /tmp/foo
4  1-2
5  3-4
6  5-6
7  sed -n 'h;n;G;s/\n/-/p' /tmp/foo
8  2-1
9  4-3
10 6-5
11 sed -r '/[3]/{h;n;G;s/(^|\n)/-/g}' /tmp/foo
12 1
13 2
14 3
15 -4-3
16 5
17 6
```

exemples

```
1  #!/bin/bash -v
2
3  sed -rne '/root/s/[ ]*(:|^) [ ]*./\U&/gp'  /etc/passwd
4  Root:X:0:0:Root:/root:/bin/bash
5  Operator:X:11:0:Operator:/root:/sbin/nologin
6  echo abcdefgh | sed 's/../\u&-/g'
7  Ab-Cd-Ef-Gh-
8
9  echo abcde | sed  's/./&\n/g;' > /tmp/abcd
10 sed = /tmp/abcd | sed -n '1~2N;s/\s/-/p'
11 1-a
12 2-b
13 3-c
14 4-d
15 5-e
```

exemples

```
1  #!/bin/bash -v
2  echo -e '1\n2\n3\n4\n5\n6\n7' > /tmp/nbr
3  sed -ne 'N;N;s/\n/:/gp'  /tmp/nbr
4  1:2:3
5  4:5:6
6  sed -e  'x; n;'    /tmp/nbr  | tr  '\n'  '/';  echo
7  /2/1/4/3/6/5/
8  sed -e '3,5d'     /tmp/nbr  | tr  '\n'  '/';  echo
9  1/2/6/7/
10 sed -ne '3,5p'    /tmp/nbr  | tr  '\n'  '/';  echo
11 3/4/5/
12 sed -ne '3,5!p'   /tmp/nbr  | tr  '\n'  '/';  echo
13 1/2/6/7/
```

s/regexp/replacement/flags

```
1  #!/bin/bash -v
2
3  sed -ne '/sh$/s/:.*:/  --> /p'  /etc/passwd
4  root  --> /bin/bash
5  pat  --> /bin/bash
6  pl  --> /bin/bash
7  juju  --> /bin/bash
8  sed -ne '/sh$/s/\([^:]*\):.*:\([^:]*\)  /\2 <-- \1/p' /etc/
       passwd
9  /bin/bash <-- root
10 /bin/bash <-- pat
11 /bin/bash <-- pl
12 /bin/bash <-- juju
```

awk

Un mini langage pour manipuler les colonnes d'un fichier. `awk` a été créé aux Bell Labs dans les années 70s par :

- Alfred Aho
- Peter Weinberger
- Brian Kernighan.

```
↝ cat /etc/passwd | awk -F: '{ print $1 }'
```

C'est un des langages qui a inspiré `perl`.

- wiki awk
- gawk
- voir aussi : `locate`, `xargs`

awk

```
↝ awk '/^p.*l/ {print $1}' /etc/passwd
pulse:x:998:996:PulseAudio
pl:x:1000:1000:pl:/home/pl:/bin/bash
↝ awk -F: '/^p.*l/ {print $2+$3+$4}' /etc/passwd
1994
2000
↝ awk --posix -F ':' '$4 ~ /5[0-9]{2}/ { print
    $1, $4}' /etc/passwd
pl 500
pat 501
```

Perl combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. Larry Wall

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

```
grep -E '(sh|home)' | awk -F: '{print $1}'
```

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

```
grep -E '(sh|home)' | awk -F: '{print $1}'
```

```
awk -F: '/home/ && /sh/ {print $1}' /etc/passwd
```

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

```
grep -E '(sh|home)' | awk -F: '{print $1}'
```

```
awk -F: '/home/ && /sh/ {print $1}' /etc/passwd
```

```
awk -F: '$6 ~ /home/ && $7 ~ /sh/ {print $1}' /
    etc/passwd
```

awkword

```
lynx --dump $1 | awk '
    BEGIN { FS="[^a-zA-Z]+" }

    {
        for (i=1; i<=NF; i++) {
            word = tolower($i)
            words[word]++
        }
    }

    END {
        for (w in words)
            printf("%3d %s\n", words[w], w)
    }
' | grep -E '[a-z]{3}' | sort -rn | head -12
```

vi

L'éditeur visuel `vi` (William Joy, BSD) est un bon laboratoire pour pratiquer les expressions régulières utiles pour l'éditeur non visuel `sed`.

```
⇝ vi test.txt
```

Vous comprendrez que `vi` démarre en mode commande et qu'il fonctionne principalement en trois modes :

- insertion
- commande
- visuel

CTRL-c bascule en mode commande. CTRL-v bascule en mode visuel

vimbook

commande

commande	action	exemple
:help	aide	:help yank
:w	sauvegarde	:w file.tex
:q	quitte	
:q !	quitte sans sauver	
:wq	modifie et quitte	
:sh	lancer un shell	:sh ... `exit`
:make	lancer `make`	
:ab	abréviation	:ab anti anticonstitutionnellement
:split	nouveau cadre	:split common.c

bouger

commande	commentaire	portée
w	suivant	mot
b	précédent	mot
^	début	ligne
0	début	ligne
$	fin	ligne
:0	début fichier	
:5	ligne	fichier
:$	fin	fichier
CTRL-W	changer de cadre	

rechercher

commande	commentaire
:/*motif*	rechercher le motif
n	suivant
N	précédent

insérer

commande	action	commentaire
i	insérer	avant le curseur
a		aprés le curseur
A		fin de ligne
I		début de ligne
o		ligne suivante
O		ligne précédente
:r *file*	un fichier	:r foo.txt
:r !*commande*	une sortie	:r ! `wc` bar.txt

diverse

commande	action	commentaire
u	annuler	undo
U	annuler	undo line
.	répéter	
m	marquer	bloc a, b
CTRL-V	marquer	visuel
J	concaténer	des lignes
~	casser	lettre
g~w	casser	mot
gU$	majusculer	curseur-EOL
guw	minusculer	mot
CTRL-A	incrémenter	nombre
CTRL-X	décrémenter	nombre

couper

commande	portée
x	caractère
5x	5 caractères
dw	mot
5dw	5 mots
dd	ligne
5dd	5 lignes
D	curseur-EOL
d´a	marque a
:5,7d	lignes 5,6,7

copier

commande	commentaire
yw	copier un mot
y3w	3 mots
yy	une ligne
y7	7 lignes
y'z	la marque z
:10,20y	les lignes 10-20.
:'<,'>y	le bloc visuel.

coller

commande	action	commentaire
p	coller	sous le curseur
P	coller	au dessus
:10,20t 100	copier	lignes 10-20 vers 100
:10,20m 100	deplacer	lignes 10-20
:100,$m 50	deplacer	100-EOF vers 50.

remplacer

commande	action	portée
r	remplacer	caractère
cw	changer	mot
cc	changer	ligne
C	changer	curseur-EOL

substituer

commande	action	portée
:%s/motif/nouveau/g	substituer	fichier
:%s/motif/nouveau		une fois par ligne
:%s/motif/nouveau/gc	cofirmation	fichier
:10,20s/motif/nouveau/g		10 a 20.
:s/.*/U&/	masjusculer	ligne courante.
:s/motif/nouveau&/g		fichier
:/motif/-2,+4d	efface	contexte
:s/[ABC]/[abc]/g	A devient a	B devient

action contextuelle

commande :g/chaine/commande	action
:g/target/s/0/ZERO/g	remplace 0 par ZERO dans les targets
:g/^[ ]*$/d	supprime toutes les lignes vides
:g/target/t7	les lignes cibles en 7
:g/target/cd 80	centre les lignes ciblées
:g !/target/d	efface les lignes
:v/target/d	idem
:g/re/p	as `grep` !

oxford dictionary