ANNÉE 2003-2004 SESSION DE MAI 2004

Licence Sciences Technologies

GU : Licence Sciences Technologies (semestre 2) **UE :** INF103
Épreuve de : Utilisation des systèmes informatiques
Date : 12 mai 2004 - Saint Achille **Durée :** 1h30
Documents manuscripts autorisés
Épreuve de Monsieur Delmas

Répondre dans les cadres directement sur le sujet ou cocher lorsque nécessaire la ou les bonnes réponses – ATTENTION : **cocher une mauvaise réponse retire un point**

Indiquer votre identificateur d'anonymat :

⋆ Sous les systèmes du types Unix, quel est le login d'usage du super utilisateur ?

admin ☐ administrator ☐
superuser ☐ super-user ☐
Root ☐ root ☐
Obiwan Kenobi ☐ secret-user ☐

⋆ Cochez les systèmes d'exploitation qui sont du type Unix.

Linux ☐ Sun/Solaris ☐
Windows NT ☐ MacOS X ☐
NetBSD ☐ FreeBSD ☐

⋆ Sur les systèmes de type Unix citez :

3 programmes clients de lecture du courrier électronique	3 programmes clients de lecture des forums de discussion	3 programmes clients de navigation web
–	–	–
–	–	–
–	–	–

⋆ Quel est le nom du fournisseur d'accès Internet de l'Université Bordeaux 1 ?

free ☐ renater ☐
aol ☐ reaumur ☐
tiscaly ☐ wanadoo ☐

⋆ Quel est le nom du fournisseur d'accès Internet de tout l'enseignement supérieur et de la recherche en France ?

free ☐ aol ☐
renater ☐ reaumur ☐
tiscaly ☐ france telecom ☐

⋆ Cochez les affirmations vraies : le super utilisateur peut,

lire le contenu de tous les fichiers utilisateurs ☐ changer le mot de passe des utilisateurs ☐
traverser tous les répertoires ☐ changer les droits des fichiers des utilisateurs ☐
connaître le mot de passe des utilisateurs ☐ retrouver un fichier utilisateur effacé par mégarde ☐
changer les liens de parenté entre processus ☐ changer un compte utilisateur en super utilisateur ☐

⋆ L'utilisateur login-1 du groupe unix etu exécute dans son *shell* les commandes suivantes :

```
$ cd
$ chmod 755 .
$ mkdir bar
$ chmod 732 bar
$ cat > bar/foo.txt
Hello, world.
^d
$
```

Cochez les commandes pouvant être effectuées sans erreur par un utuilisateur login-2 du groupe unix etu.

- ☐ `$ ls -al ~login-1/bar`
- ☐ `$ ls ~login-1/bar/foo.txt`
- ☐ `$ cd ~login-1/bar`
- ☐ `$ rm ~login-1/bar/foo.txt`
- ☐ `$ cat > ~login-1/bar/foo2.txt`
`do not panic`
`^d`
- ☐ `$ mkdir ~login-1/bar/projet`

⋆ Écrire le résultat produit par la suite de commandes :

```
$ cd ; pwd
/net/home/login
$ cd ..; (cd ~/..; (cd .././../.; pwd) ; pwd ); pwd
```

⋆ Écrire le résultat produit par la suite de commandes :

```
$ cd ; pwd
/net/home/login
$ FOO=pwd
$ N="cd ..; (cd ~/..; (cd .././../.; pwd); eval $FOO); pwd"
$ eval $N; eval $N; eval $N; eval $N; pwd
```

⋆ Écrire le résultat produit par la suite de commandes :

```
$ cd ; pwd
/net/home/login
$ N="cd ..; (cd ~/..; (cd .././../.; pwd) ; pwd ); pwd"
$ eval $N; (eval $N; (eval $N; (eval $N))); pwd
```

⋆ Que produit l'exécution des commandes suivantes

```
$ ls /bin
ar  as  apl cc cak  cp  dd  echo  gawk  gzip  ln  mknod  netstat
$ N='/bin/[a-c]?'
$ M=N
$ echo $N
```

$ echo "$M"

⋆ Que produit l'exécution des commandes suivantes

```
$ UN=un
$ sh
$ echo /$UN/
```

```
$ ^d
$
$ ( export DEUX=deux ); export TROIS=trois
$ bash
$ export QUATRE=4
$ echo /$UN/$DEUX/$TROIS/$QUATRE/
```

```
$ ^d
$ echo $QUATRE
```

⋆ Que produit l'exécution des commandes suivantes

```
$ echo "main (int argc, char **argv) { exit(EXIT_SUCCESS); }" | tr -cs [A-Za-z0-9] '\n'
```

⋆ Écrire un script-shell statistique.sh qui se comporte comme un filtre et qui affiche sur la sortie standard la fréquence d'apparition (classé par ordre décroissant) de chaque mot ou nombre provenant du flux d'entrée standard. Par exemple,

```
$ cat essai.c
main (int argc, char **argv)
{
  printf("Hello, world - Hello everybody\n");
  exit(0);
}
$ ./statistique.sh < essai.c
      2 Hello
      1 world
      1 printf
      1 n
      1 main
      1 int
      1 exit
      1 everybody
      1 char
      1 argv
      1 argc
      1 0
$
```

⋆ Écrire un script-shell change-droits-acces-repertoires.sh qui prend en paramètre un nom de répertoire et une chaine de caractères correspondant à des droits d'accès. Ce script doit positionner les droits d'accès (fourni par le second paramètre) sur le répertoire (fourni par le premier paramètre) ainsi que tous ces sous-répertoires, le script ne devra afficher aucun message d'erreur. Le script devra fonctionner même s'il y a des répertoire commançant par le caractère point.

⋆ Expliquer ce qui différencie un programme exécutable d'un processus.

⋆ Quel est l'ordre de création des processus lorsque l'on tape une commande du type :

```
$ processus-1 | processus-2 | processus-3
```

1 puis 2 puis 3 ..... ☐ 1, 2 et 3 sont créés en même temps ..... ☐
3 puis 2 puis 1 ..... ☐ 2 puis 1 puis 3 ..... ☐

⋆ Comment peut-on passer un processus en avant-plan dans l'état STOP ?

Control-z ..... ☐ Escape ..... ☐
Control-c ..... ☐ Control-d ..... ☐

⋆ Qu'effectuera la commande bg dans le contexte suivant ?

```
$ jobs
[1]   Stopped                 xdaliclock
[2]-  Stopped                 xclock
[3]+  Stopped                 ls -R / | grep cp
[4]   Running                 xeyes &
$ bg
```

xdaliclock est exécuté en avant-plan ..... ☐
xclock est executé en arrière-plan ..... ☐
ls -R / est executé en arrière-plan ..... ☐
xeyes passe en avant-plan ..... ☐

ls -R / | grep cp est executé en arrière-plan ..... ☐
grep cp est exécuté en avant-plan ..... ☐
ls -R / est executé en avant-plan ..... ☐
xeyes passe dans l'état *Stopped* ..... ☐

⋆ Quel est le PID du noyau d'un système de type UNIX ?

0 ..... ☐
1 ..... ☐
2 ..... ☐
3 ..... ☐

⋆ Le script-shell mystere.sh prend 2 paramètres, un entier suivi d'un nom de fichier.

```
$ cat mystere.sh
#!/bin/bash

if [[ ( $(wc -l < $2) -lt $1 ) || ( $1 -le 0 ) ]]
then
 cat $2
else
 cat $2  | head -$(expr $1 - 1)
 cat $2  | tail -$(expr $(wc -l < $2) - $1)
fi
$
```

En supposant que le fichier poeme.txt contienne

```
$ cat poeme.txt
Pourquoi faut-il que nos rêves
Même les plus petits
S'arrêtent comme des manèges
Sans que l'on en ait envie ?
$
```

Que produira la commande

```
$ ./mystere.sh 2 poeme.txt; ./mystere.sh 22 poeme.txt
```

⋆ Le script-shell mkdir-cd.sh a pour but de créer un nouveau répertoire et de placer l'utilisateur directement dans ce répertoire

```
$ cat mkdir-cd.sh
#!/bin/bash

mkdir $1
cd $1
$
```

Donner un exemple d'utilisation de ce script mkdir-cd.sh sous votre shell afin de créer et de vous placer directement dans le répertoire ∼/foo (on supposera que ce répertoire n'existait pas).

⋆ Écrire une autre solution permettant à l'utilisateur d'utiliser une commande mkdir-cd mais qui n'utilise pas de script-shell.

⋆ Gestion d'un annuaire personnel

Un annuaire est un fichier contenant un *item* par ligne. Chaque ligne contient une suite de champs fournissant différentes informations sur une personne physique ou morale. Cet annuaire peut contenir des lignes relatives à vos amis, à des relations professionnels, des entreprises ou autres. Ainsi, chaque ligne de votre annuaire commencera par une information d'identification (apelé type) du contenu de la ligne. Vous trouverez en annexe A un exemple d'un tel annuaire (fichier mon-annuaire.txt). Dans cet exemple (cf. annexe A) vous pouvez constater, qu'une ligne peut appartenir à plusieurs types différents. Par exemple, Alberto Jean-Louis est visiblement à la fois dans votre annuaire professionnel et personnel.

La structure de l'annuaire est simple. Les champs sont séparés par le caractère ';'. Un champs débute toujours par un nom de champ suivit du caractère ':'. Un champ peut contenir différente valeur séparée par le caractère ','. On supposera que les valeurs des champs ne peuvent pas contenir de caractères ',' ou '|' ou ':' ou ';'.

On supposera que dans le répertoire contenant le fichier mon-annuaire.txt se trouvent également plusieurs *script-shell* (supposés avoir les bons droits d'exécution) permettant d'effectuer différents traitements d'un fichier annuaire. Ces *script-shell* sont présentés en annexe B.

⋆ Quel est le résultat de la commande

```
$ ./script-1.sh -f mon-annuaire.txt WWW
```

⋆ Quel est le résultat de la commande

```
$ ./script-2.sh -f mon-annuaire.txt www
```

⋆ Quel est le résultat de la commande

```
$ ./script-2.sh -f mon-annuaire.txt "borDeaux    cedex"
```

⋆ Quel est le résultat de la commande

```
$ ./script-3.sh -f mon-annuaire.txt -champ
```

⋆ Quel est le résultat de la commande

```
$ ./script-4.sh -f mon-annuaire.txt -d -champ fax -type entreprise
```

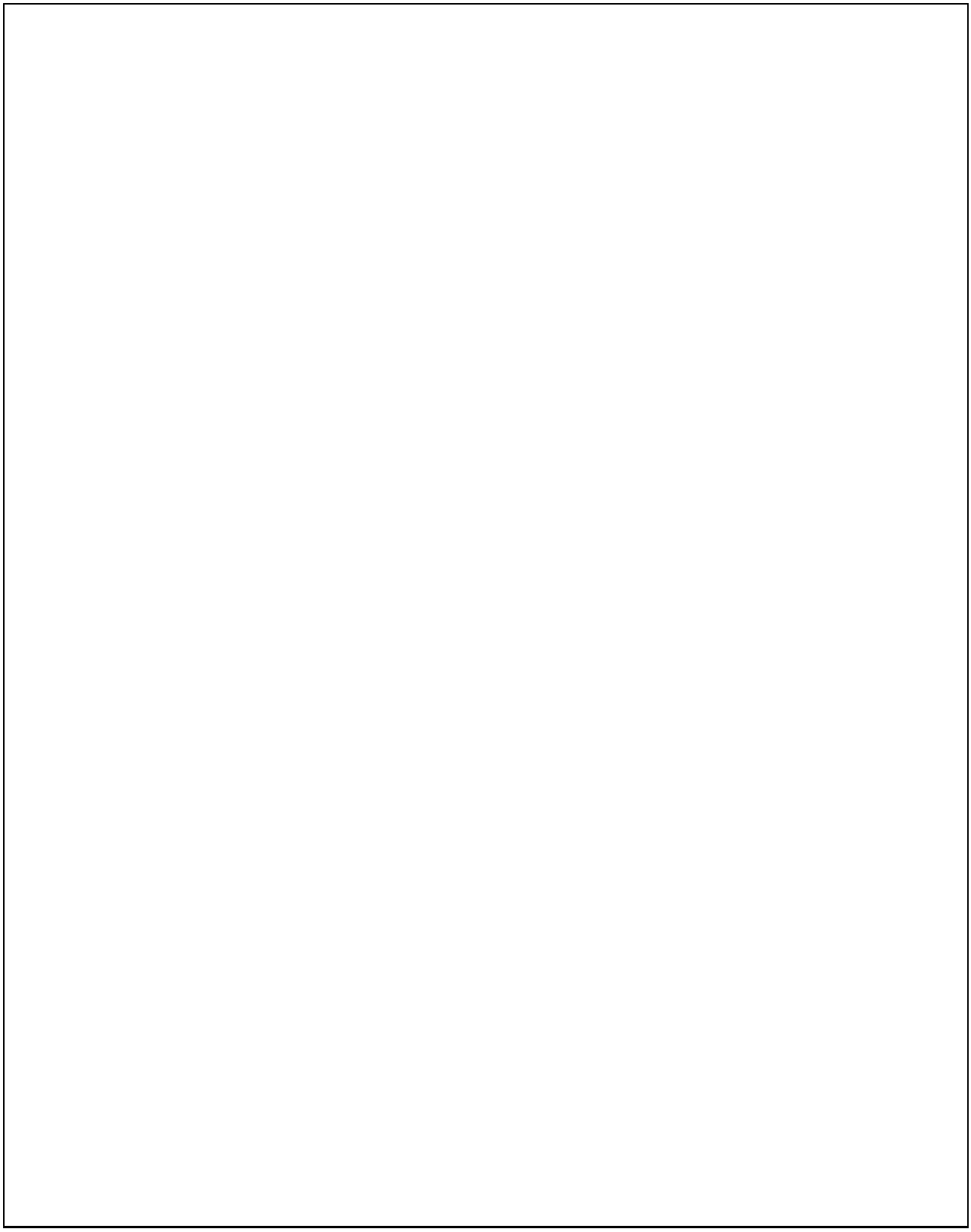
⋆ Écrire un script shell supprime-lignes.sh qui prend en premier paramètre un nom de fichier puis une liste d'entier. Ce script doit renvoyer sur la sortie standard le fichier le contenu du fichier passé en premier paramètre auquel on aura supprimé les lignes passée dans les autres paramètres. Par exemple,

`$ ./supprime-ligne ./mon-annuaire.txt 2 5 11` devra renvoyer le contenu du fichier mon-annuaire.txt supprimé des lignes 2, 5 et 11.

Votre script devra supporté n'importe quel ordre de passage des numéro de ligne. Par exemple,
`$ ./supprime-ligne ./mon-annuaire.txt 11 2 5` aura le même effet.

Votre script devra également prendre en compte des intervalles de lignes via la notation ≺*numéro-début-ligne*≻-≺*numéro-fin-ligne*≻ (avec numero-debut-ligne toujours inférieur ou égal à numero-fin-ligne). Par exemple
`$ ./supprime-ligne ./mon-annuaire.txt 3-5 1` aura pour but de supprimer les lignes n° 1, 3, 4 et 5.

Si un numéro de ligne passé en paramètre n'existe pas dans le fichier, cela n'affectera pas le contenu du fichier renvoyé par le script. Par contre votre script devra effectuer les vérifications nécessaire d'existence du fichier ainsi que la vérification des types (entier positif, ou intervalle) passé en paramètre après le nom du fichier. Votre script devra renvoyer un message d'usage sur la sortie d'erreur standard et positionner une valeur de retour adéquate.



FIN.

Annexes

A Structure d’un fichier annuaire

```
$ cat mon-annuaire.txt
type:perso;nom:Durand, Alain Jean-Marie;06 43 23 67 12 (portable);adresse:123 allée des acacias, 24567 TORINA
type:pro;nom:Becker, Guillaume;adresse:44eme rue, Z7K6 New-York, USA;email:gbecker@design-int.org;tel:899 345 776 (fixe)
type:perso,pro;nom:Alberto,   Jean-Louis;adresse:2 rue Camus, 33000 Bordeaux       Cedex;email:dfk@free.uk, ttl@world.net
type:entreprise;nom:fnac;adresse:50 rue Sainte Catherine 33001 BORDEAUX CEDEX;email:bordeaux@fnac.tm.fr;tel:05 56 00 22 10
type:entreprise;nom:restaurant Le   CamUs;tel: 0 567 778 899;web:www.resto-le-camus.fr;fax:05 65 44 33 22;email:le-camus@restos.fr
type:pro,entreprise;nom:Le  barrAl;tel: 0 567 778 898;web:WWW.resto-le-barral.fr;fax:05 65 44 33 21;email:le-barral@restos.fr
$
```

B Script-shell de traitement d’un fichier annuaire

B.1 script-1.sh

```
$ cat script-1.sh
#!/bin/bash

USAGE="echo $(basename $0) -f fichier motif-simple; exit 1"
USAGE_PB_FICHIER_EN_ENTREE="echo Le fichier $2 n\’exite pas ou est vide ou non lisible; exit 2"

# test la validité des arguments
if [[ $# != 3 || $1 != -f ]]
then
  eval $USAGE
fi

# test la validité du fichier d’entrée
test -f $2 && test -r $2 && test -s $2 || eval $USAGE_PB_FICHIER_EN_ENTREE

ANNUAIRE=$2
MOTIF=$3

cat $ANNUAIRE | grep $MOTIF
$
```

B.2 script-2.sh

```
$ cat script-2.sh
#!/bin/bash

USAGE="echo $(basename $0) -f fichier motif-simple; exit 1"
USAGE_PB_FICHIER_EN_ENTREE="echo Le fichier $2 n\’exite pas ou est vide ou non lisible; exit 2"

# test la validité des arguments
if [[ $# != 3 || $1 != -f ]]
then
  eval $USAGE
fi

# test la validité du fichier d’entrée
test -f $2 && test -r $2 && test -s $2 || eval $USAGE_PB_FICHIER_EN_ENTREE

ANNUAIRE=$2
MOTIF=$3

cat $ANNUAIRE | tr -s [:space:] | tr [:upper:] [:lower:] | \
 grep "$(echo $MOTIF | tr -s [:space:] | tr [:upper:] [:lower:] )"
$
```

B.3 script-3.sh

```
$ cat script-3.sh
#!/bin/bash

USAGE="echo $(basename $0) -f fichier -type\|-champ; exit 1"
USAGE_PB_FICHIER_EN_ENTREE="echo Le fichier $2 n\’exite pas ou est vide ou non lisible; exit 2"

# test la validité des arguments
if [[ $# != 3 || $1 != -f ]]
then
```

```
  eval $USAGE
fi

# test la validité du fichier d'entrée
test -f $2 && test -r $2 && test -s $2 || eval $USAGE_PB_FICHIER_EN_ENTREE

ANNUAIRE=$2

case $3 in
  -type)
        cat $ANNUAIRE | cut -d ":" -f2 | cut -d ";" -f1 | tr "," "\n" | sort | uniq -c | cut -f2
        ;;
  -champ)
         FICHIER_TMP=tmpfile.tmp
         exec 3<$ANNUAIRE
         exec 5>$FICHIER_TMP

         ( while read ligne
           do
             IFS=";"
             set $ligne
             for i in $*
             do
               echo $i | cut -d ":" -f1
             done
            done
         ) <&3 >&5

         cat $FICHIER_TMP | sort | uniq -c | cut -f2
         rm -f $FICHIER_TMP
        ;;
   *)
      eval $USAGE
esac
$
```

B.4 script-4.sh

```
$ cat script-4.sh
#!/bin/bash

# -- USAGE et test de validité du nombre d'arguments
USAGE="echo $(basename $0) -f fichier -d/-rd [-o annuaire-en-sortie] \
       -champ nom-champ -type type-annuaire; exit 1"
USAGE_PB_FICHIER_EN_ENTREE="echo Le fichier $2 n\'exite pas ou est vide \
                            ou non lisible; exit 2"

  # test la validité des arguments
if [[ ( $# != 7 && $# != 9 ) || $1 != -f ]]
then
  eval $USAGE
fi

  # test la validité du fichier d'entrée
test -f $2 && test -r $2 && test -s $2 || eval $USAGE_PB_FICHIER_EN_ENTREE

ANNUAIRE_EN_SORTIE=/dev/stdout

ANNUAIRE=$2
```

```
shift; shift

while [ ""$1 != "" ]
do
  case $1 in
    -d)
       ORDRE=""
       shift
       ;;
    -rd)
       ORDRE="-r" # l'option -r est utiliser par la commande sort pour inverser (reverse)
       shift      # le résultat des comparaisons
       ;;
    -o)
       shift
       ANNUAIRE_EN_SORTIE=$1
       shift
       ;;
    -champ)
       shift
       CHAMP=$1
       shift
       ;;
    -type)
       shift
       TYPE=$1
       shift
       ;;
  esac
done

COMMANDE="cat $ANNUAIRE | grep -e \"^type:.*${TYPE}[,;]\" | \
  sed -e 's/;${CHAMP}:/|/g '| sort $ORDRE -t '|' -k 2 | sed -e 's/|/;${CHAMP}:/g '"

eval $COMMANDE > $ANNUAIRE_EN_SORTIE
```

- La commande `sed -e 's/;${CHAMP}:/|/g '` est un filtre. Ce filtre remplace (après la phase d'évaluation des variables) la chaîne de caractères `;${CHAMP}:` par le caractère `|`
- La commande `sed -e 's/|/;${CHAMP}:/g '` fait l'inverse, c'est-à-dire qu'elle remplace (après la phase d'évaluation des variables) le caractère `|` par la chaîne de caractères `;${CHAMP}:`

C Extraits de Pages du manuel

C.1 sort

```
SORT(1)                          FSF                          SORT(1)



NAME
       sort - sort lines of text files

SYNOPSIS
       sort [OPTION]... [FILE]...

DESCRIPTION
       Write sorted concatenation of all FILE(s) to standard out
       put.
```

```
Ordering options:

-b, --ignore-leading-blanks ignore leading blanks

-d, --dictionary-order
       consider only blanks and alphanumeric characters

-f, --ignore-case
       fold lower case to upper case characters

-g, --general-numeric-sort
       compare according to general numerical value

-i, --ignore-nonprinting
       consider only printable characters

-M, --month-sort
       compare (unknown) < ‘JAN’ < ... < ‘DEC’

-n, --numeric-sort
       compare according to string numerical value

-r, --reverse
       reverse the result of comparisons

Other options:

-c, --check
       check whether input is sorted; do not sort

-k, --key=POS1[,POS2]
       start a key at POS1, end it at POS 2 (origin 1)

-m, --merge
       merge already sorted files; do not sort

-o, --output=FILE
       write result to FILE instead of standard output

-s, --stable
       stabilize sort by disabling last-resort comparison

-S, --buffer-size=SIZE
       use SIZE for main memory buffer

-t, --field-separator=SEP  use  SEP  instead  of  non-  to
       whitespace transition

-T, --temporary-directory=DIR
       use DIR for temporaries, not $TMPDIR or /tmp multi
       ple options specify multiple directories

-u, --unique
       with -c: check for strict ordering otherwise:  out
       put only the first of an equal run

-z, --zero-terminated
       end lines with 0 byte, not newline
```

```
       +POS1 [-POS2]
              start  a key at POS1, end it before POS2 (origin 0)
              Warning: this option is obsolescent

       --help display this help and exit

       --version
              output version information and exit

       POS is F[.C][OPTS], where F is the field number and C  the
       character  position  in  the  field, both counted from one
       with -k, from zero with the  obsolescent  form.   OPTS  is
       made  up  of  one  or more single-letter ordering options,
       which override global ordering options for that  key.   If
       no key is given, use the entire line as the key.

       SIZE  may be followed by the following multiplicative suf
       fixes: % 1% of memory, b 1, k 1024 (default),  and  so  on
       for M, G, T, P, E, Z, Y.

       With no FILE, or when FILE is -, read standard input.

       ***  WARNING  ***  The locale specified by the environment
       affects sort order.  Set LC_ALL=C to get  the  traditional
       sort order that uses native byte values.
[...]
```

C.2 uniq

```
UNIQ(1)                       FSF                          UNIQ(1)



NAME
       uniq - remove duplicate lines from a sorted file

SYNOPSIS
       uniq [OPTION]... [INPUT [OUTPUT]]

DESCRIPTION
       Discard  all  but  one  of successive identical lines from
       INPUT (or standard input), writing to OUTPUT (or  standard
       output).

       -c, --count
              prefix lines by the number of occurrences

       -d, --repeated
              only print duplicate lines

       -D, --all-repeated
              print all duplicate lines

       -f, --skip-fields=N
              avoid comparing the first N fields

       -i, --ignore-case
              ignore differences in case when comparing
```

```
       -s, --skip-chars=N
              avoid comparing the first N characters

       -u, --unique
              only print unique lines

       -w, --check-chars=N
              compare no more than N characters in lines

       -N     same as -f N

       +N     same as -s N (obsolescent; will be withdrawn)

       --help display this help and exit

       --version
              output version information and exit

       A  field is a run of whitespace, then non-whitespace char
       acters.  Fields are skipped before chars.
[...]
```

C.3 cut

```
CUT(1)                        FSF                           CUT(1)



NAME
       cut - remove sections from each line of files

SYNOPSIS
       cut [OPTION]... [FILE]...

DESCRIPTION
       Print  selected  parts of lines from each FILE to standard
       output.

       -b, --bytes=LIST
              output only these bytes

       -c, --characters=LIST
              output only these characters

       -d, --delimiter=DELIM
              use DELIM instead of TAB for field delimiter

       -f, --fields=LIST
              output only these fields;  also print any line that
              contains  no  delimiter  character,  unless  the -s
              option is specified

       -n     (ignored)

       -s, --only-delimited
              do not print lines not containing delimiters

       --output-delimiter=STRING
              use STRING as the output delimiter the  default  is
```

```
              to use the input delimiter

       --help display this help and exit

       --version
              output version information and exit

       Use  one, and only one of -b, -c or -f.  Each LIST is made
       up of one range, or many ranges separated by commas.  Each
       range is one of:

       N      N'th byte, character or field, counted from 1

       N-     from N'th byte, character or field, to end of line

       N-M    from  N'th  to  M'th  (included) byte, character or
              field

       -M     from first to M'th (included)  byte,  character  or
              field

       With no FILE, or when FILE is -, read standard input.
[...]
```

C.4 tr

```
TR(1)                          FSF                           TR(1)



NAME
       tr - translate or delete characters

SYNOPSIS
       tr [OPTION]... SET1 [SET2]

DESCRIPTION
       Translate, squeeze, and/or delete characters from standard
       input, writing to standard output.

       -c, --complement
              first complement SET1

       -d, --delete
              delete characters in SET1, do not translate

       -s, --squeeze-repeats
              replace sequence of characters with one

       -t, --truncate-set1
              first truncate SET1 to length of SET2

       --help display this help and exit

       --version
              output version information and exit

       SETs are specified as strings of characters.  Most  repre
       sent themselves.  Interpreted sequences are:
```

```
\NNN   character  with  octal value NNN (1 to 3 octal dig
       its)

\\     backslash

\a     audible BEL

\b     backspace

\f     form feed

\n     new line

\r     return

\t     horizontal tab

\v     vertical tab

CHAR1-CHAR2
       all characters from CHAR1  to  CHAR2  in  ascending
       order

[CHAR*]
       in SET2, copies of CHAR until length of SET1

[CHAR*REPEAT]
       REPEAT  copies  of  CHAR,  REPEAT octal if starting
       with 0

[:alnum:]
       all letters and digits

[:alpha:]
       all letters

[:blank:]
       all horizontal whitespace

[:cntrl:]
       all control characters

[:digit:]
       all digits

[:graph:]
       all printable characters, not including space

[:lower:]
       all lower case letters

[:print:]
       all printable characters, including space

[:punct:]
       all punctuation characters

[:space:]
       all horizontal or vertical whitespace
```

```
       [:upper:]
              all upper case letters

       [:xdigit:]
              all hexadecimal digits

       [=CHAR=]
              all characters which are equivalent to CHAR

       Translation occurs if -d is not given and  both  SET1  and
       SET2  appear.  -t may be used only when translating.  SET2
       is extended to length of SET1 by repeating its last  char
       acter   as  necessary.   Excess  characters  of  SET2  are
       ignored.  Only [:lower:] and [:upper:] are  guaranteed  to
       expand in ascending order; used in SET2 while translating,
       they may only be used in pairs to specify case conversion.
       -s uses SET1 if not translating nor deleting; else squeez
       ing uses SET2 and occurs after translation or deletion.
[...]
```

C.5 wc

```
WC(1)                          FSF                           WC(1)



NAME
       wc - print the number of bytes, words, and lines in files

SYNOPSIS
       wc [OPTION]... [FILE]...

DESCRIPTION
       Print  newline, word, and byte counts for each FILE, and a
       total line if more than one FILE is  specified.   With  no
       FILE, or when FILE is -, read standard input.

       -c, --bytes
              print the byte counts

       -m, --chars
              print the character counts

       -l, --lines
              print the newline counts

       -L, --max-line-length
              print the length of the longest line

       -w, --words
              print the word counts

       --help display this help and exit

       --version
              output version information and exit
[...]
```

C.6 egrep

```
GREP(1)                                                    GREP(1)



NAME
       grep, egrep, fgrep - print lines matching a pattern

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

DESCRIPTION
       Grep  searches the named input FILEs (or standard input if
       no files are named, or the file name - is given) for lines
       containing a match to the given PATTERN.  By default, grep
       prints the matching lines.

       In addition, two variant  programs  egrep  and  fgrep  are
       available.   Egrep  is  the same as grep -E.  Fgrep is the
       same as grep -F.

OPTIONS
       -A NUM, --after-context=NUM
              Print NUM lines of trailing context after  matching
              lines.

       -a, --text
              Process  a  binary file as if it were text; this is
              equivalent to the --binary-files=text option.

       -B NUM, --before-context=NUM
              Print NUM lines of leading context before  matching
              lines.

       -C [NUM], -NUM, --context[=NUM]
              Print NUM lines (default 2) of output context.

       -b, --byte-offset
              Print  the byte offset within the input file before
              each line of output.

       --binary-files=TYPE
              If the first few bytes of a file indicate that  the
              file  contains binary data, assume that the file is
              of type TYPE.  By default, TYPE is binary, and grep
              normally  outputs  either a one-line message saying
              that a binary file matches, or no message if  there
              is  no  match.   If  TYPE  is  without-match,  grep
              assumes that a binary file does not match; this  is
              equivalent to the -I option.  If TYPE is text, grep
              processes a binary file as if it were text; this is
              equivalent   to   the  -a  option.   Warning:  grep
              --binary-files=text might  output  binary  garbage,
              which  can have nasty side effects if the output is
              a terminal and if the  terminal  driver  interprets
              some of it as commands.

       -c, --count
```

Suppress normal output; instead print a count of matching lines for each input file. With the -v, --invert-match option (see below), count non-matching lines.

-d ACTION, --directories=ACTION
If an input file is a directory, use ACTION to process it. By default, ACTION is read, which means that directories are read just as if they were ordinary files. If ACTION is skip, directories are silently skipped. If ACTION is recurse, grep reads all files under each directory, recursively; this is equivalent to the -r option.

-E, --extended-regexp
Interpret PATTERN as an extended regular expression (see below).

-e PATTERN, --regexp=PATTERN
Use PATTERN as the pattern; useful to protect patterns beginning with -.

-F, --fixed-strings
Interpret PATTERN as a list of fixed strings, separated by newlines, any of which is to be matched.

-f FILE, --file=FILE
Obtain patterns from FILE, one per line. The empty file contains zero patterns, and therefore matches nothing.

-G, --basic-regexp
Interpret PATTERN as a basic regular expression (see below). This is the default.

-H, --with-filename
Print the filename for each match.

-h, --no-filename
Suppress the prefixing of filenames on output when multiple files are searched.

--help Output a brief help message.

-I Process a binary file as if it did not contain matching data; this is equivalent to the --binary-files=without-match option.

-i, --ignore-case
Ignore case distinctions in both the PATTERN and the input files.

-L, --files-without-match
Suppress normal output; instead print the name of each input file from which no output would normally have been printed. The scanning will stop on the first match.

-l, --files-with-matches

Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match.

--mmap If possible, use the mmap(2) system call to read input, instead of the default read(2) system call. In some situations, --mmap yields better perfor mance. However, --mmap can cause undefined behav ior (including core dumps) if an input file shrinks while grep is operating, or if an I/O error occurs.

-n, --line-number
Prefix each line of output with the line number within its input file.

-q, --quiet, --silent
Quiet; suppress normal output. The scanning will stop on the first match. Also see the -s or --no-messages option below.

-r, --recursive
Read all files under each directory, recursively; this is equivalent to the -d recurse option.

-s, --no-messages
Suppress error messages about nonexistent or unreadable files. Portability note: unlike GNU grep, traditional grep did not conform to POSIX.2, because traditional grep lacked a -q option and its -s option behaved like GNU grep's -q option. Shell scripts intended to be portable to traditional grep should avoid both -q and -s and should redirect output to /dev/null instead.

-U, --binary
Treat the file(s) as binary. By default, under MS-DOS and MS-Windows, grep guesses the file type by looking at the contents of the first 32KB read from the file. If grep decides the file is a text file, it strips the CR characters from the original file contents (to make regular expressions with ^ and $ work correctly). Specifying -U overrules this guesswork, causing all files to be read and passed to the matching mechanism verbatim; if the file is a text file with CR/LF pairs at the end of each line, this will cause some regular expressions to fail. This option has no effect on platforms other than MS-DOS and MS-Windows.

-u, --unix-byte-offsets
Report Unix-style byte offsets. This switch causes grep to report byte offsets as if the file were Unix-style text file, i.e. with CR characters stripped off. This will produce results identical to running grep on a Unix machine. This option has no effect unless -b option is also used; it has no effect on platforms other than MS-DOS and MS-Win dows.

```
[...]

       -v, --invert-match
              Invert the sense of matching, to select  non-match
              ing lines.

       -w, --word-regexp
              Select  only  those  lines  containing matches that
              form whole words.  The test is  that  the  matching
              substring  must  either  be at the beginning of the
              line, or preceded by a non-word constituent charac
              ter.   Similarly,  it  must be either at the end of
              the line or  followed  by  a  non-word  constituent
              character.   Word-constituent  characters  are let
              ters, digits, and the underscore.

       -x, --line-regexp
              Select only those matches that  exactly  match  the
              whole line.

       -y     Obsolete synonym for -i.

       -Z, --null
              Output  a  zero  byte  (the  ASCII  NUL  character)
              instead of the character that  normally  follows  a
              file  name.   For  example, grep -lZ outputs a zero
              byte after each file name instead of the usual new
              line.   This  option  makes the output unambiguous,
              even in  the  presence  of  file  names  containing
              unusual  characters like newlines.  This option can
              be used with commands like find -print0,  perl  -0,
              sort  -z,  and  xargs  -0 to process arbitrary file
              names, even those that contain newline  characters.
[...]
```