Sample Answers

Question 1

How to get the current directory into the prompt? For example, when your current directory is now `/home/csa0/bin`, let prompt be like

`fkokus07@csa0:/home/csa0/bin/%`

and when you `cd` to `/usr/lib`, let prompt be like

`fkokus07@csa0:/usr/lib/%`

Answer

Actually it is a little bit difficult. You may at first think of a setting like

```
% set prompt = ’‘whoami‘@‘hostname‘:‘pwd‘%’
```

but it will fail, because the command substitution of `‘pwd‘` is performed only once when a shell read the `.cshrc` (or `.login`) file at the starting up time. The concept for the implementation is to divide a prompt setting into two procedure:

1. Change the current directory to somewhere, and
2. Change the prompt sign including the directory name there.

One of the typical examples is shown below.

```
alias setprompt ’set prompt="‘whoami‘@‘hostname‘:${cwd}%"’
setprompt
alias cd ’chdir \!* && setprompt’
```

Only you have to write down above three lines into your `.cshrc` file. The third line is the actual alias for `cd` which at first changes directory, and if it succeeds, sets a prompt sign by `setprompt` alias. Content of `setprompt` is defined before. The middle line is necessary to set the initial prompt. Try what will happen if you omit the second line.

Question 2

What should we do to remove the files below? They can not be deleted in the ordinary ways. In addition, how to make the files which have such extraordinary names?

- `-abc` (which starts with hyphen)
- `?abc` (which starts with wildcard `?`)
- `*abc` (which starts with wildcard `*`)
- `abc def` (which contains blank within)

`  abc` (which starts with blank)
`;` (whose name is “colon”)
`-;` (whose name is “hyphen + colon”)

Answer

You cannot delete files whose name contain metacharacters of shell in the ordinary ways. In these cases you have to use relative/absolute referencing, or quotations of shell in order to deny the meaning of metacharacters.

If a filename contains the wildcards (like `?abc` or `*abc`), you have to quote them in quotations. For example

```
% rm '?abc'
% rm '*abc'
```

or

```
% rm "?abc"
% rm "*abc"
```

or using backslash,

```
% rm \?abc
% rm \*abc
```

It is also the same when a filename contains blank (white space), like

```
% rm 'abc def'
% rm ' abc'
```

or

```
% rm abc\ def
% rm \ abc
```

It is also the same when a filename begins with semicolon, like

```
% rm ';'
```

or

```
% rm \ ;
```

When a filename begins with hyphen (like `-abc`) it is a little bit confusing because a hyphen is not a metacharacter, so the quotations can not deny its meaning. In that case the filename is interpreted as one of the options for `rm` command, so you have to specify the filename with the relative referencing such as

```
% rm ./-abc
```

In this way you can tell your shell that the hyphen does not denote a option of `rm` command.

When a filename starts with hyphen and semicolon, you have to use both relative referencing and quoting. For example, you can delete a file named `-;` like

```
% rm ./-';'
```

You can create these file in the same way by using a command `touch` and so on.

Question 3

Someone would like to print out the contents of `fileA` with line numbers, and did the following command, but failed.

```
% cat -n fileA > lpr
```

Why?, and what is the correct way?

Answer

This is an easy mistake. `lpr` is interpreted as a executable command only when it is located at the top of the command line, or just after the pipe sign `|`. In the above case, `> lpr` means for shell to create a new file named `lpr`, so he couldn't send his data to printer. The correct way is of course

```
% cat -n fileA | lpr
```

Question 4

To concatenate `fileA` and `fileB`, someone tried the command

```
% cat fileA fileB >> fileA
```

In this case, what kind of output did he get? And why is that? Then, What is the best way to achieve his object?

Answer

Suppose content of `fileA` and `fileB` are as follows.

```
% cat fileA
We are in fileA.
% cat fileB
I am now in fileB.
```

When you enter the above command, shell will return a warning like

```
cat:  input fileA is output
```

This warning means that the input file of command `cat` is also used for the output of redirection `>`. And the result will be somewhat difficult from what you wanted:

```
% cat fileA
I am now in fileB.
```

`fileA` is now identical with `fileB`.

The reason is as follows. When you specify `cat fileA`, shell try to open `fileA` in the read-only mode. Also, When you specify `> fileA`, shell try to open `fileA` in the overwriting mode, which contradicts the appointment of read-only mode. Due to this inconsistency, the content of `fileA` is once cleared out, and then the command

```
% cat empty_file fileB >> fileA
```

is performed. That is why `fileA` becomes identical with `fileB`.

The best way to achieve his object is to use a temporal file. For example, following procedures like

```
% cat fileA fileB > temp__file__
% mv temp__file__ fileA
```

will satisfy his object. Of course in this case, the original content of `fileA` will be lost.

Question 5

On the machines in this classroom, someone who wanted to know the settings of his `prompt` did

```
% echo $prompt
```

However, the shell returned an error message

```
echo: No match.
```

and failed. Similar commands like `echo $history` works. Why is that? What is the best way to show the setting of `prompt` successfully?

Answer

See where your prompt is set. You will find it in `.cshrc` like

```
set prompt = "‘whoami‘@‘hostname‘[\!]%"
```

which is for example substituted by shell like

```
fkokus08@cse8[!]%
```

Notice that the exclamation `!` will not be substituted because of the negation by preceding `\` here. So, when you type above command

```
% echo $prompt
```

it is expanded to

```
fkokus08@cse8[!]%
```

Here, shell interprets `!]\%` as a *history event* expression, and searches the string which matches with it. And when shell failed to search any matching strings, he give a waning

```
echo: No match.
```

and exits. One of the best way to show the setting of `prompt` successfully is to quote the variable `$prompt` in double quotation in order to deny the meaning of metacharacter `!`, like

```
% echo "$prompt"
```

Question 6

Alias to `pwd` is perhaps `echo $cwd` in your `.cshrc` file. However, alias format like

```
alias pwd echo $cwd
```

does not give the right output. Why? What is the right way?

Answer

All aliases are set when shell consults `.cshrc` file when it starts up. So, if a setting of `pwd` is like

```
alias pwd echo $cwd
```

shell sets alias for `pwd` to `echo` *cwd (where you were in when shell started).* For example, since shell usually starts up from your home directory (here we put it `/home/csa0/you`), above setting will produce the results

```
% pwd
/home/csa0/you
```

wherever directory you are in. The right way to set an alias for `pwd` is again to deny the meaning of metacharacter `$` in `$cwd` not to be substituted (expanded) when `.cshrc` is consulted, using double quotations such as

```
alias pwd 'echo $cwd'
```

This method produces the right result.

Question 7

In SunOS system (which you are using now), how long can words on C shell be? Can you set a variable which has 1,000,000,000 characters long?

Answer

It is clearly stated at the section of LIMITATIONS in the on-line manual of `csh` as

> LIMITATIONS
>
> *Words can be no longer than 1024 characters.* The system limits argument lists to 1,048,576 characters. However, the maximum number of arguments to a command for which filename expansion applies is 1706. Command substitutions may expand to no more characters than are allowed in the argument list. To detect looping, the shell restricts the number of alias substitutions on a single line to 20.

So the answer is **1024** characters. Of course this value differs from each operating system and hardware.

Question 8

Redirection method of `>&` mixes the standard output and standard error output. Suppose there is a program which creates both, how can we separate them by using `>` and `>&`? Sample code is as below. Using this code (write, save, compile and execute), try to separate the outputs into standard output (first `fprintf`) and standard error output (second `fprintf`).

```
#include <stdio.h>
main(){
    fprintf(stdout, "This line is for standard output.\n");
```

```
    fprintf(stderr, "And this line is for standard err output.\n");
}
```

Answer

The concept is as follows:

1. Subtract only the standard output from the mixed output.
2. Then, the remnant consists of only the standard error output.

For example, actual implementation is (suppose you named the above executable code as `bothout`) as follows:

```
% (bothout > file.stdout) >& file.stderr
```

An important point here is make the first redirection `bothout > file.stdout` be braced with `()` so that shell will interpret it as a single command. Otherwise, if you do such like

```
% bothout > file.stdout >& file.stderr
```

shell will give an error message

```
Ambiguous output redirect.
```

end exits.

Question 9

Editor `Emacs` makes backup files which start from `#` or trailing `~` (like `#file` or `file~`). Someone wanted to delete all these files at a time, and made a C shell script like

```
#!/bin/csh
rm -f #* *~
```

However, it didn't work. Why is this?, and how you can solve the problem?

Answer

This is rather simple. Remember that the sharp sign (`#`) in the shell scripts denotes the begging of comments. So in the second line, `* *~` is completely neglected because they are located *after* `#` sign, and shell interprets it as

```
rm -f
```

and exits without doing anything. To solve the problem, you have to deny the meaning of metacharacter `#` in the above script such as

```
#! /bin/csh
rm -f \#* *~
```

Question 10

In MS–DOS system, it is possible to rename all the `*.foo` files to `*.bar` at a time by using the command `ren`. But UNIX equivalent `mv` can not work in this way. For example,

```
% mv *.foo *.bar
```

will fail. Why is that? Then, try to make a shell script which rename all the `*.foo` files to `*.bar` at a time such like

```
% somescript *.foo *.bar
```

or

```
% somescript foo bar
```

Answer

Why doesn't `mv *.foo *.bar` work? Think about how the shell expands wildcards. `*.foo` and `*.bar` are expanded *before* the `mv` command ever sees the arguments. Depending on your shell, this can fail in a couple of ways. C shell prints `No match.` because it can't match `*.bar`. Bourne shell executes `mv a.foo b.foo c.foo *.bar`, which will only succeed if you happen to have a single directory named `*.bar`, which is very unlikely and almost certainly not what you had in mind.

You can do it with a loop to `mv` each file individually. C shells has its own variable substitution feature, so you can use simple loops like:

```
#! /bin/csh -x
foreach f ( *.foo )
    mv $f $f:r.bar
end
```

Or if you want you specify the extensions from your command line, use `$argv[]` like

```
#! /bin/csh -x
set ext1 = $argv[1]
set ext2 = $argv[2]
foreach file ( *.${ext1} )
```

```
    mv $file $file:r.${ext2}
end
```

You can simply execute the script by specifying both extensions as

```
% script foo bar
```

Question 11

Below, a C shell script to exchange the two file names is shown. This script does not work if each specified file is a directory file. Rewrite the script and make it handle the directory files also.

```
#! /bin/csh
if ($#argv != 2) then
  set cmd = $0
  echo "Usage: $cmd:t [file1] [file2]"
  exit
else
  if      (! -e $argv[1]) then
    echo "$argv[1] does not exist."
    exit
  else if (! -e $argv[2]) then
    echo "$argv[2] does not exist."
    exit
  else
    set TMPFILE = /tmp/NAME1_____
    /bin/cp    $argv[1] $TMPFILE
    /bin/mv -f $argv[2] $argv[1]
    /bin/mv -f $TMPFILE $argv[2]
    /bin/rm -f $TMPFILE
  endif
endif
```

Answer

`mv` command will not move a directory from one file system to another. So in this case, you have to change your current directory to the parent of the object directory. To implement this, commands `basename` and `dirname` are available.

- `basename`
  `basename` deletes any prefix ending in `/` and the suffix, if present in string. It directs the

result to the standard output, and is normally used inside substitution marks (‘ ‘) within shell procedures.

- `dirname`
  `dirname` delivers all but the last level of the path name in string.

Of course you can implement the directory exchanging by using `-r` or `-R` option of `cp` command. According to the on-line manual of `cp`,

> `-r`
> `-R`
>
> Recursive. If any of the source files are directories, copy the directory along with its files (including any subdirectories and their files); the destination must be a directory.

Below is one of the examples of the script which we use `basename` and `dirname` commands. Try to write down and execute it.

```
#! /bin/csh
if ($#argv != 2) then
  set cmd = $0
  echo "Usage: $cmd:t [file1] [file2]"
  exit
else
  if      (! -e $argv[1]) then
    echo "$argv[1] does not exist."
    exit
  else if (! -e $argv[2]) then
    echo "$argv[2] does not exist."
    exit
  else
    if ( -d $argv[1] && -d $argv[2] ) then
      #
      # case of directory
      #
      set TMPDIR = __TMPDIR__
      pushd ‘dirname $argv[1]‘
        /bin/mv -f $argv[1]:t $TMPDIR
      popd
      pushd ‘dirname $argv[2]‘
        /bin/mv -f $argv[2]:t $argv[1]:t
      popd
      pushd ‘dirname $argv[1]‘
        /bin/mv -f $TMPDIR $argv[2]:t
      popd
```

```
    else if ( -f $argv[1] && -f $argv[2] ) then
      #
      # case of file
      #
      set TMPFILE = /tmp/NAME1_____
      /bin/cp    $argv[1] $TMPFILE
      /bin/mv -f $argv[2] $argv[1]
      /bin/mv -f $TMPFILE $argv[2]
      /bin/rm -f $TMPFILE
    else
        echo "Cannot exchange file and directory."
    endif
  endif
endif
```

Question 12

Below, a C shell script to calculate the total bytes in subdirectories recursively is shown. At the top of the script, the C shell variable `nonomatch` is set. What for? (hint: try to make an empty subdirectory, and run the script)

```
#! /bin/csh
set nonomatch
if ($#argv != 1) then
  echo "usage: $0 directory"
  exit
endif
set count = 0
foreach file ($1/*)
  if      (-f "$file") then
    set size = `/bin/ls -l  "$file"`
  else if (-d "$file") then
    set size = `/bin/ls -ld "$file"`
    set recurse = `$0 "$file"`
    @ count = $count + $recurse
  else
    echo "$file not included in the total" >>! /tmp/notintotal
    continue
  endif
  @ count = $count + $size[4]
end
```

```
echo $count
```

Answer

When you unset `nonomatch` in the above script, perhaps an error message like

```
foreach: No match.
@: Expression syntax.
```

will appear and the script terminates abnormally. In general, to debug a C shell script, you can specify `-x` option such as

```
#! /bin/csh -x
....
```

which means to set the `echo` variable; echo commands after all substitutions and just before execution. You can then see the implementation of command substitutions visually.

As in *hints*, try to make an empty subdirectory. Here we suppose it is named `empty/`. Then run the script (here we call it `total`) by

```
% total empty
```

and see what will happen. Probably, logs like

```
if ( 1 != 1 ) then
set count = 0
foreach file ( empty/* )
foreach: No match.
```

will be shown. What does this mean?

Shell will try to match all the files to a variable `$file` at the line

```
foreach file ($1/*)
```

in the above script, and expands asterisk `*`. However, in this case `$1` is substituted with an empty directory, so there is no file in `$1`; *i.e.* shell can not match any filenames to `$file`. That is the reason why shell returns an error like

```
foreach file ( empty/* )
foreach: No match.
```

By setting `nonomatch` in the script, shell won't expand asterisk `*`. In the case of an empty directory, shell simply skip it and there will occur no error.

Question 13

Make a shell script which rings the terminal bell, such like
`% bell` (rings once)
`% bell 5` (rings five times)

Answer

These are easy to implement. Escape sequence (control codes) for terminal bell is `^G` (control + G), so the simplest example with no argument is

```
#! /bin/csh
echo -n ^G
```

Option `-n` of `echo` means not to add a NEWLINE to the output. If you want to make the script understand the command line argument, one of the examples becomes such like below.

```
#! /bin/csh
if ($#argv < 1) then
  set i = 1
else if ($#argv > 1) then
  echo "usage: $0 [number]"
  exit
else
  set i = $argv[1]
endif
while ($i >= 1)
  echo -n ^G
  sleep 1
  @ i--
end
```

Try to write them down and execute.

Question 14

What should we do to get the list of all files which satisfy the three conditions

1. which are under `/home`, and
2. which are modified within these two weeks, and
3. which have the octal permissions of 644 (`-rw-r--r--`).

by **one line command**? (hint: use the command `find`)

Answer

The exact answer is:

```
find /home -mtime -14 -perm 644 -print
```

`find` is a very powerful command which recursively descends the directory hierarchy for each pathname in the pathname list, seeking files that match a logical expression written using the operators. About the detail you should consult the on-line manual or textbooks. Briefly explain, above condition

1. is satisfied by the path list `/home`
2. is satisfied by the option `-mtime -14`
3. is satisfied by the option `-perm 644`

Likewise, if you want to find all files which satisfy the conditions

1. which is under current directory
2. which has not been accessed during these three weeks
3. which has the octal permissions of 755 (`-rwxr-xr-x`).
4. which is an ordinary file

you can achieve your object by simply enter a command

```
find . -type f -perm 755 -atime +21 -print
```

Question 15

What way are there to know whether the user to whom you sent an E-mail has read your mail or not?

Answer

Simply you can make it by `finger` command. If you run the command and get following output like

```
% finger tito
Login name: tito                        In real life: Takashi ITO
Directory: /home/pluto/tito             Shell: /bin/csh
New mail received Tue Jan 10 18:31:56 1995;
  unread since Tue Jan 10 18:26:00 1995
```

then he has new mails unread. In the case of following output like

```
% finger tito
Login name: tito                        In real life: Takashi ITO
Directory: /home/pluto/tito             Shell: /bin/csh
No unread mail
```

then he doesn't have any unread mails, so it's likely for him to have already read your mail. For more detailed part, consult the on-line manual of `finger` command.

Question 16

Make aliases as follows:

1. Sort the results of `ps -aux` in order of process ID (`PID`).
2. Sort the results of `ps -aux` in order of the CPU time used by each process.
3. Show *your* processes only in the results of `ps -aux`.

Answer

You have to become familiar with `sort` and `grep` command. Since both of them are quite major and popular commands, there will be detailed explanations about them in your textbooks.

The exact method to implement 1 is for example

```
% alias pss '/bin/ps -aux | sort -n +1'
```

Then a result will be sort in order of `PID` as

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
root         0  0.0  0.0    0    0 ?  D    Jan  5  0:26 swapper
root         1  0.0  0.1   52   24 ?  S    Jan  5  0:01 /sbin/init -
root         2  0.0  0.0    0    0 ?  D    Jan  5  0:00 pagedaemon
root        51  0.0  0.0   72    0 ?  IW   Jan  5  0:08 portmap
root        56  0.0  0.0   40    0 ?  IW   Jan  5  0:00 keyserv
hiroshi   1506  0.0  0.0   72    0 p3 IW   Jan  6  0:00 -sh (csh)
....

hiroshi  13814  0.0  0.0   72    0 p7 IW   Jan  9  0:00 -sh (csh)
hiroshi  13838  0.0  0.0  308    0 p7 IW   Jan  9  0:00 -csh (tcsh)
root     15680  0.0  0.0   36    0 ?  IW   18:07   0:03 in.rlogind
root     15783  0.0  0.1   36   28 ?  S    18:18   0:04 in.rlogind
root     15852  0.0  0.9   24  208 ?  S    18:39   0:00 in.comsat
```

You have to use option +*number* of `sort` to specify the sort field in your data. The option `-n` means

```
-n

    Numeric collating sequence.  An initial numeric string, consisting of optional
    blanks, optional minus signs, and zero or more digits with an optional decimal
    point, is sorted by arithmetic value.
```

according to the on-line manual.

Similarly, the exact method to implement 2 is for example

```
% alias psc '/bin/ps -aux | sort -n +2'
```

Then a result will be sort in order of `%CPU` as

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
hiroshi    641  0.0  0.0   68    0 co IW   Jan  5  0:00 -csh (csh)
hiroshi   1493  0.0  0.0   28    0 co IW   Jan  6  0:00 /usr/X11R5/bin/xinit
....

root     15783  0.0  0.1   36   28 ?  S    18:18   0:07 in.rlogind
wnn        419  0.0  0.0 2064    0 ?  IW   Jan  5  0:01 jserver
tito     15902  7.7  1.1   60  248 p1 S    18:46   0:00 emacs
tito     15901 30.8  2.1  220  476 p1 R    18:46   0:00 od -a filetestch
```

Implementation of alias 3 needs help of `grep`. An exact method is for example

```
% alias psy '/bin/ps -aux | sort -n +1 | grep `whoami`'
```

Here it is better to specify your login name by `` `whoami` `` in case you have different login names in each machine.

Question 17

Try to write one of the UNIX commands `cat` by yourself (simple version is enough) using C language or shell script. Send the code (or script) by E-mail.

Answer

The possible answers vary very much. For example, one of the simplest examples using `putc` function is as follows:

```
#include <stdio.h>
```

```
main(argc, argv)
int argc;
char *argv[];
{
    FILE *fin;
    int c;

    if ((fin=fopen(argv[1], "r")) == NULL){
        perror(argv[1]);
        exit(1);
    }
    else {
        while ((c=getc(fin)) != EOF){
            putc(c, stdout);
        }
        fclose(fin);
    }
}
```

What this code does is quite simple,

1. Read a character one by one from the input file, and
2. Display it on-screen (standard output) one by one whatever it is.

Of course it cannot accept any command line options. When you want to concatenate several files at a time, combine above C code with below C shell script (`mycat` to be a name of above C program)

```
#! /bin/csh
while ($#argv > 0)
  mycat $argv[1]
  shift
end
```

You can well simulate `/bin/cat` by this script.

For reference, a actual source code of `cat` is shown below. This is a version which is now used in a free UNIX system (BSD/386). Try to read and run it if you want.

```
/*
 * Copyright (c) 1989 The Regents of the University of California.
 * All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by
 * Kevin Fall.
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef lint
char copyright[] =
"@(#) Copyright (c) 1989 The Regents of the University of California.\n\
 All rights reserved.\n";
#endif /* not lint */

#ifndef lint
static char sccsid[] = "@(#)cat.c       5.15 (Berkeley) 5/23/91";
#endif /* not lint */

#include <sys/param.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int bflag, eflag, nflag, sflag, tflag, vflag;
int rval;
char *filename;

void cook_args(), cook_buf(), raw_args(), raw_cat();
void err __P((int, const char *, ...));

main(argc, argv)
        int argc;
        char **argv;
{
        extern int optind;
        int ch;

        while ((ch = getopt(argc, argv, "benstuv")) != EOF)
                switch (ch) {
                case 'b':
                        bflag = nflag = 1;      /* -b implies -n */
                        break;
                case 'e':
                        eflag = vflag = 1;      /* -e implies -v */
                        break;
                case 'n':
                        nflag = 1;
                        break;
                case 's':
                        sflag = 1;
                        break;
                case 't':
                        tflag = vflag = 1;      /* -t implies -v */
                        break;
                case 'u':
                        setbuf(stdout, (char *)NULL);
                        break;
```

```
                case 'v':
                        vflag = 1;
                        break;
                case '?':
                        (void)fprintf(stderr,
                            "usage: cat [-benstuv] [-] [file ...]\n");
                        exit(1);
                }
        argv += optind;

        if (bflag || eflag || nflag || sflag || tflag || vflag)
                cook_args(argv);
        else
                raw_args(argv);
        if (fclose(stdout))
                err(1, "stdout: %s", strerror(errno));
        exit(rval);
}

void
cook_args(argv)
        char **argv;
{
        register FILE *fp;

        fp = stdin;
        filename = "stdin";
        do {
                if (*argv) {
                        if (!strcmp(*argv, "-"))
                                fp = stdin;
                        else if (!(fp = fopen(*argv, "r"))) {
                                err(0, "%s: %s", *argv, strerror(errno));
                                ++argv;
                                continue;
                        }
                        filename = *argv++;
                }
                cook_buf(fp);
                if (fp != stdin)
                        (void)fclose(fp);
        } while (*argv);
```

```
}

void
cook_buf(fp)
        register FILE *fp;
{
        register int ch, gobble, line, prev;

        line = gobble = 0;
        for (prev = '\n'; (ch = getc(fp)) != EOF; prev = ch) {
                if (prev == '\n') {
                        if (ch == '\n') {
                                if (sflag) {
                                        if (!gobble && putchar(ch) == EOF)
                                                break;
                                        gobble = 1;
                                        continue;
                                }
                                if (nflag && !bflag) {
                                      (void)fprintf(stdout, "%6d\t", ++line);
                                      if (ferror(stdout))
                                              break;
                                }
                        } else if (nflag) {
                                (void)fprintf(stdout, "%6d\t", ++line);
                                if (ferror(stdout))
                                        break;
                        }
                }
                gobble = 0;
                if (ch == '\n') {
                        if (eflag)
                                if (putchar('$') == EOF)
                                        break;
                } else if (ch == '\t') {
                        if (tflag) {
                              if (putchar('^') == EOF || putchar('I') == EOF)
                                      break;
                              continue;
                        }
                } else if (vflag) {
                        if (!isascii(ch)) {
```

```
                                if (putchar('M') == EOF || putchar('-') == EOF)
                                        break;
                                ch = toascii(ch);
                        }
                        if (iscntrl(ch)) {
                                if (putchar('^') == EOF ||
                                    putchar(ch == '\177' ? '?' :
                                    ch | 0100) == EOF)
                                        break;
                                continue;
                        }
                }
                if (putchar(ch) == EOF)
                        break;
        }
        if (ferror(fp)) {
                err(0, "%s: %s", strerror(errno));
                clearerr(fp);
        }
        if (ferror(stdout))
                err(1, "stdout: %s", strerror(errno));
}

void
raw_args(argv)
        char **argv;
{
        register int fd;

        fd = fileno(stdin);
        filename = "stdin";
        do {
                if (*argv) {
                        if (!strcmp(*argv, "-"))
                                fd = fileno(stdin);
                        else if ((fd = open(*argv, O_RDONLY, 0)) < 0) {
                                err(0, "%s: %s", *argv, strerror(errno));
                                ++argv;
                                continue;
                        }
                        filename = *argv++;
                }
```

```
                raw_cat(fd);
                if (fd != fileno(stdin))
                        (void)close(fd);
        } while (*argv);
}

void
raw_cat(rfd)
        register int rfd;
{
        register int nr, nw, off, wfd;
        static int bsize;
        static char *buf;
        struct stat sbuf;

        wfd = fileno(stdout);
        if (!buf) {
                if (fstat(wfd, &sbuf))
                        err(1, "%s: %s", filename, strerror(errno));
                bsize = MAX(sbuf.st_blksize, 1024);
                if (!(buf = malloc((u_int)bsize)))
                        err(1, "%s", strerror(errno));
        }
        while ((nr = read(rfd, buf, bsize)) > 0)
                for (off = 0; off < nr; nr -= nw, off += nw)
                        if ((nw = write(wfd, buf + off, nr)) < 0)
                                err(1, "stdout");
        if (nr < 0)
                err(0, "%s: %s", filename, strerror(errno));
}

#if __STDC__
#include <stdarg.h>
#else
#include <varargs.h>
#endif

void
#if __STDC__
err(int ex, const char *fmt, ...)
#else
err(ex, fmt, va_alist)
```

```
        int ex;
        char *fmt;
        va_dcl
#endif
{
        va_list ap;
#if __STDC__
        va_start(ap, fmt);
#else
        va_start(ap);
#endif
        (void)fprintf(stderr, "cat: ");
        (void)vfprintf(stderr, fmt, ap);
        va_end(ap);
        (void)fprintf(stderr, "\n");
        if (ex)
                exit(1);
        rval = 1;
}
```

Question 18

Tell at least two ways to get a recursive directory listing.

Answer

This is very simple. One of the following may do what you want:

```
% ls -R
```

Notice that not all versions of `ls` have a option `-R` which has a meaning of recursively listing. Or

```
% find . -print
```

This should should work everywhere. Or else,

```
% du -a .
```

shows you both the name and size of the current directory.

Question 19

Explain the reason that the command `chmod` is available only for the owner of the files and superuser.

Answer

Of course it is for security in the filesystems. If `chmod` is available for any users in the system, anyone can make your unreadable files readable, like

```
% chmod a+r *
```

so your security (or privacy) in the filesystems will be completely broken.

Question 20

Explain briefly the major differences between MS–DOS and UNIX, advantages and disadvantages of them.

Answer

The answer is probably written on your textbooks, so we hardly mention here. For example,

- MS–DOS is a single tasking, single user operating system. On the other hand, UNIX is a multitasking operating system and has a multitasking capability.
- MS–DOS can available mainly on the IBM compatible personal computers, but UNIX can available on almost any kinds of computers from PCs, workstations, mainframes, and even supercomputers.
- ...

Question 21

Explain briefly the advantages and the disadvantages of E-mail compared with the other communication methods (such as telephone, facimilie, ...).

Answer

Probably you can find the answer in your textbooks, so we omit here.

Question 22

On UNIX editors like `vi` or `emacs`, function keys or arrow keys are not available. Why is that?

Answer

Probably you can find the answer in your textbooks, but one of the most important points here is important here is

- Editors like `vi` or Emacs was originally designed to work on any UNIX systems and on any hardwares. But there are keyboards on some machines which don't have function keys, arrow keys, and even escape keys, so `vi` or Emacs was produced not to use these special keys.

Question 23

`cal` is a command to display a calendar for the specified year. According to the on-line manual of `cal`, they say "Try September 1752". So, try to do that, and explain briefly what the result means. It is not joking (I think).

Answer

Actually, I don't exactly know the reason. But someone in this class told me that it may due to a historical and religious reason. For a more detailed part, please ask for someone who is familiar with the western history.