SECTION 1   INTRODUCTION TO UNIX

1.0   INTRODUCTION

This section is intended to introduce you to the Unix operating system. It will provide you with a basic understanding of the Unix operating system, its file and directory structure. We have also explained the architecture and components of Unix, further the section contains different useful Unix commands, those are explained in detail with the help of examples.

1.1   OBJECTIVES

After completing this lab manual, you should be able to:

* get basic understanding of the Unix operating system;

* understand file and directory structure of Unix;

* know the basic Unix commands, and

* know how to get Unix help;

3l2   OVERVIEW OF·UNIX

Even after thirty-five years of its creation Unix is still regarded as one of the most versatile, flexible and powerful operating systems in the computer world. Before you start swimming in your Unix shell, you must find out why people still regard it as powerful. As you may know, it was created at Bell Labs in 1970, written in the C Programming Language, which was developed at the same time. It supports a large number of simultaneous users, runs with few alterations on many hardware platforms (provides some platform independence), and of course it was and is, a simple, elegant, and easy to use (at least compared to its predecessors) operating system. In the early 1980s, the two strands of Unix development – AT&T and Berkeley – continued in parallel. The Berkeley strand got a major boost from Sun Microsystems, which used the Berkeley code as the basis for its Sun OS operating system.

This section is only a brief introduction to Unix operating system and does not include information on how to use all of its capabilities. Let's see the historical development of Unix at a glance:

* Kenneth Thompson, Dennis Ritchie, and others at AT&T Bell Labs developed first Unix version in 1969-1970.
* The above system was rewritten in the programming language C in 1972-1973.
* The seventh version (V7) of Unix was released in 1979.

During these 35 years of its development, different flavors of Unix system have evolved. Some of these Unix flavors are given on following Table I with the name of the organization which participated in that development.

Table 1: List of Unix Flavors

Unix flavor	Organization name
AIX	IBM
FreeBSD	FreeBSD Group
HP-UX	Hewlett-Packard Company
Irix	Silicon Graphics, Inc.
Linux	Several groups
MacOS X Server	Apple Computer, Inc.
NetBSD	NetBSD Group
OpenBSD	OpenBSD Group
OpenLinux	Caldera Systems, Inc.
Red Hat Linux	Red Hat Software, Inc.
Reliant Unix	Siemens AG
SCO Unix	The Santa Cruz Operation Inc.
Solaris	Sun Microsystems
SuSE	S.u.S.E., Inc.

Due to, Unix is multi-user and multi-tasking environment, its flexibility and portability, facilities like electronic mail and the numerous programming, text processing and scientific utilities available, Unix became popular amongst the scientific and academic communities.

Basic Unix Elements

The six basic elements of Unix are:

1) Commands

2) Files

3) Directories

4) Environment

5) Processes

6) Jobs

1) **Commands** are the instructions you give to the system to inform it what it is to do.

2) **Files** are collections of data. A file is similar to a container in which you can store documents or raw facts and figures, which are stored in directories.

3) **Directory** is similar to a file basket that contains man, files. A directory can also contain other directories.

4) **Environment** is a collection of different items that explain or modify how your computing session will be carried out.

5) **Process** is a command or application running on a computer.

6) **Job** is the sequence of instructions given to a computer from the time to initiate a particular task. A job may have one or more processes in it.

In our next discussion we will use and explore these elements in detail but first you need to understand different components of the Unix operating system.

Unix Components

The Unix operating system is made up of three components–the kernel, the shell and the programs (applications). As we have earlier defined the program, so here we will only define the kernel and the shell.

The Kernel

The kernel is the core of the Unix operating system. Its main respdnsibilities are to allocate time and memory to programs, file management and communications in response to system calls.

The interface between the use; and the kernel is called the shell. When a user logs in, the login program checks the **username** and password, and then starts another program **called** the shell. The shell is a command line interpreter (CLI). It interprets the **commands** the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt (% on our systems). Shell and kernel work together, for example a user types a command, which has the effect of removing some file. The shell searches the file store for the file containing the command, and then requests the kernel, through system calls, to execute a command on file. When the process has finished running, the shell then returns the Unix prompt to the user, indicating that it is waiting for further commands.

Unix System Architecture

Broadly specify, the Unix **architecture** is divided into three layers–Application Program, System Calls and Kernel. As we can see in Figure *1*, the layers of Unix operating system are:

Application Programs Created by Users		
Shells editor and commands (who, we, grep, **comp**)		
Compilers and system libraries		
System call interface to the kernel		
Signals terminal handling character I/O system terminal drivers	File system swapping block I/O system disk & tape drivers	CPU scheduling pages replacement demand paging visual memory
Kernel interface to Hardware		
Terminal controllers terminals	Device controllers disks and tapes	Memory Controllers Physical memory

Figure 1: Unix System Architecture

Everything belbw the system call interface and above the physical hardware is the Kernel. The Kernel provides the file system, CPU Scheduling, memory management and other operating system functions through system calls. Programs such as shell (Sh) and editors (vi) shown in the top layer interact with the Kernel by invoking a well-defined set of system calls. The system calls instruct the Kernel to do various operations for the calling programs and exchange data between the Kernel and the ˉ program. System calls for Unix can be roughly grouped into three categories: file manipulation, process control and information manipulation. Another category can be considered for device manipulation, but since devices in Unix are treated as (special) files, the same system calls support both files and devices.

Files and Processes

Everything in Unix is either a file or a process. In Unix, Process is identified by a unique number called process identifier (PID). They are created by users using text eclitors, running compilers etc. and must have a file name according to Unix rules. Following are the few examples of files in Unix.

- A document.
- A program written in some high-level programming language.
- A collection of binary digits
- A directory.

The Directory Structure

Let's see how files are grouped together in the directory structure in Unix. The files are arranged in a hierarchical structure, like an inverted tree as shown in *Figure* 2. The top of the hierarchy is called the root.
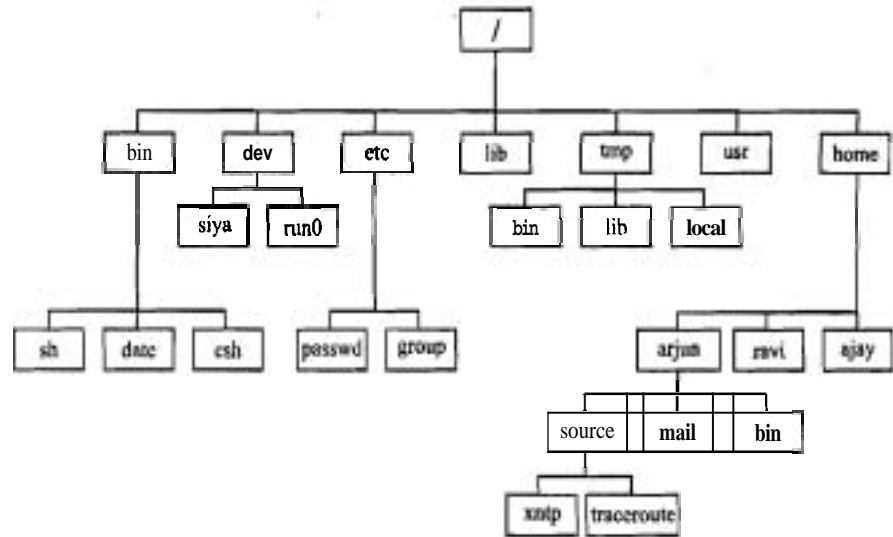

Figure 2: Directory structure of Unix

1.3   UNIX COMMANDS

When you start using Unix first you interact with a command interpreter program called the shell. Generally, Unix has two different shells C and Bourne shell. The shell you will find here in this manual is the C shell. The C shell command prompt ends with the percent sign (%). Another shell is the Bourne shell, which ends with a dollar sign ($).

Unix shells have their commands, syntax and control constructs. You can use these commands, syntax and control constructs to make your processing more efficient, or to perform repetitive tasks. In addition to this, you can store shell commands in a file (which is called a shell script) and run it as a simple program.

Commands

Using different commands in Unix you can manipulate files, data and environment. In this section we have explained the general syntax of Unix commands so that you can start working on it. A Unix command line consists of the name of the command followed by its arguments and ends with a RETURN. The general syntax for a command is:

Command Name [-flag options] *filename* or expression

You should follow these rules with Unix commands:
1)   Commands are case-sensitive.
2)   Most Commands are in lowercase.
3)   Commands must be entered at the shell prompt.
4)   Command lines must end with a RETURN.
5)   Options generally begin with a "-" (minus sign).

List

When you first log in to the Unix environment, the directory in which you enter is your home directory. The simple Is (list) command lists the contents of your current working directory. There may be no files visible in your home directory, in which case, the Unix prompt will be returned. List command does not show all the files in your home directory; it shows only those files whose name does not start with a dot

(.). What are these dot files? Files beginning with a dot (.) are known as hidden files and generally contain important information.

- To list all files in your home directory including those whose names begin with a dot.

%Is-a

Is is an example of a command which can take options: -a is a case of an option. The options change the performance of the command. There are online manual pages (man) that inform you which options a particular command can take, and how each option modifies the performance of the command. Later on, we will discuss man pages in detail. Let see a few options of list command:

Is -a   lists all the contents of the current directory, including files with initial periods, which are not usually listed.

Is -l   lists the contents of the current directory in long format, including file permissions, size, and date information.

Is -s   lists contents and file size in kilobytes of the current directory.

Working with directories

As you know directories are similar to files in Unix, in this section you will learn how to create or change the directories. Also, we will know about different special directories in Unix.

Make directory

Let's make a subdirectory in your home directory to hold the files you will be using and creating in this course. Use the following comments:

?   To make a subdirectory called netprogs in your current working directory

% mkdir netprogs

?   To see the directory netprogs which just you have just created

% ls

Change directory

The command **cd** directory means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are working in:

?   To change to the directory you have just made, type

% cd netprogs.

Special directories

As you can see there are two special directories called (.) and (..) in the netprogs directory and in all other directories in your file system.

(.) means the current directory, so typing **% cd.** will take you to current directory which is netprogms at this moment. But you should remember that there is a space between cd and the dot. Using (.) as the name of the current directory will save a lot of typing time.

(..) means the parent of the current directory, so typing **% cd ..** will take you one directory up the hierarchy (back to your home directory). Remember typing cd with no argument always returns you to your home directory. This is very useful if you are lost in the file system.

Print working directory

Pathnames enable you to work out where you are in relation to the whole file-system. For example,

? To find out the absolute **pathname** of your home directory, type *cd* to get

back to your home directory and then type

% pwd

The full **pathname** will look something like this - /a/fservb/fservb/fservb22/ gd/ hd which means that hd (your home directory) is in the directory gd (the group directory), which is located on the fservb file-server.

Understanding pathnames

What is a pathname? To understand the actual meaning of pathnames, let's takes an example where you want to see the contents of directory backups but currently you are in some other directory. Normally you can write the command as given below:

? To list the contents of your backups directory,

% Is backups

After this command you will get a message like this - ***backups: No such file or directory.***

Because a backup is not your current working directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must either *cd* to the correct directory, or specify its full pathname.

? To list the contents of your backups directory,

% Is netprogs/backups

Home directory

Home directories can also be referred to by the tilde ⁓ character. It can be used to specify paths starting at your home directory. So typing **% Is ~/netprogs** will list the contents of your netprogs directory, no matter where you are currently in the file system.

Copying Files

cp file1 file2 is the command which makes a copy of **file1** in the current working directory and calls it **file2** .

? Take a file stored in an open access area of the file system, and use the cp command to copy it **to your** netprogs directory.

% cd ~/netprogs
% cp /vol/examples/tutorial/science.txt .

Remember the dot (.) at the end of *cp* command. The dot means the current directory in Unix. The given command means copy the file *science.txt* to the current directory and keeping the same name. Assume directory **/vol/examples/tutorial/** is an area to which everyone in the class has read and copy access.

Moving/Renaming files

mv file1 file2 command is used to move a file from one place to another. This has the effect of moving rather than copying the file, so you end up with only one file rather than two. It can also be used to rename a file, by moving the file to the same directory, but giving it a different name. Assume you have a **science.bak** file

? To move the file **science.bak** to your backup directory

inside the **netprogs** directory write % mv **science.bak backups/.**

Removing files and directories

To remove a file from a directory, use the rm command. For example, we are going

? To create a copy of the **science.txt** file then delete that file.

//when you are in netprogs directory, **write//**

```
% cp science.txt tempfile.txt
% ls (to check if it has created the file)
% rm tempfile.txt
% ls (to check if it has deleted the file)
```

You can use the rmdir command to remove an empty directory.

Displaying File on Screen

Clear

? To clear the terminal window of the previous commands

% clear

This will clear all text and leave you with the % prompt at the top of the screen.

cat (concatenate)

? The command cat can be used to display the contents of a file on the screen.

% cat science.txt

Less

? To display the contents of a file onto the screen a page at a time.

% less science.txt

Press the [space-bar] if you want to see another page, type [q] if you want to quit reading.

Head

? To display the first ten lines of a file to the screen.

% head science.txt

Tail

? To display the last ten lines of a file on the screen

% tail science.txt

Searching contents using less

You can search through a text file for a keyword (pattern) using the less command. For example,

? To search through science.txt for the word 'science'

% less science.txt
/science

less finds and highlights the keyword. To search the next occurrence of the word type [n].

Searching contents using grep

grep command searches files for specified words or patterns.

? To search through science.txt for the word 'science'

% grep science science.txt

After this command you will see each line containing the word science. Always remember that the grep command is case sensitive which means it distinguishes ' between Science and science. If you want to ignore **upper/lower** case distinctions, you should use -i option like % grep -i science science.txt. To search for a pattern, you must enclose it in single quotes (the apostrophe symbol). For example,

? To search for *computer science* in science.txt

% grep -i 'computer science' science.txt

Some of the other options of grep are given below:

Options **of** grep	Meaning
-v	Display those lines that do NOT match
-n	Precede each matching line with the line number
-c	Print only the total count of matched lines

Word count

We counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines. If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters I, w, or c. For example,

? To count the words in science.txt
% wc -w science.txt

? To find out the lines in a file
% wc -l science.txt

Redirecting input and output

Most commands write to the standard output and many take their input from the standard input. When we run the cat command without specifying a file to read, it reads from the standard input (the keyboard), and on receiving the EOF (end of file) (^D), copies it to the standard output (the screen). We have option to redirect both the input and the output of commands. Let see how:

Output Redirection

We use the > symbol to redirect the output of a command. For example, first create two files listl and list2. One contains six fruit names; the other contains four fruit names.

? To join (concatenate) list1 and list2 into a new file called biglist

% cat list1 list2 > biglist

Here the system reading the contents of listl and list2 and, then the output is redirected to the **file** biglist.

Input Redirection

We use the < symbol to redirect the input of a command. The input will be accepted from a file rather than the keyboard. For example,

? To sort the list of fruits
% sort < biglist
the redirect input and output can be done together as given below in an example:

? To output the sorted list to a file
% sort < biglist > slist

Who

Who command without any with it argument, lists the login name, terminal name, and login time for each current Unix user.

? To see who is on the system with you, type
% who

Pipes

Pipes (indicated by vertical bar |) create an inter-process channel between the commands. Let's see how, if you want

? To get a sorted list of user names you can write:
% who > names.txt
% sort < names.txt

But in this case we need a temporary file called names.txt. If you want to connect the output of the who command directly to the input of the sort command. This is called as inter-process channel between two commands. The symbol for a pipe is the vertical bar |

? To get a sorted list of user names

% who | sort

? To find out how many users are logged on

% who | wc -l

Wildcards

The characters * and ? are known as wildcards. Character * will match against none or more character(s) in a file (or directory) name. For example,

? To list all files in the current directory starting with list-----

% ls list*

The character? will match exactly one character. So Is? onkey will match files like donkey and monkey.

Help Manuals

There are on-line manuals, which give information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the performance of the command. man command locates and prints the section of this manual named title in the specified chapter.

We have different options given with man command. Some of these are given below:

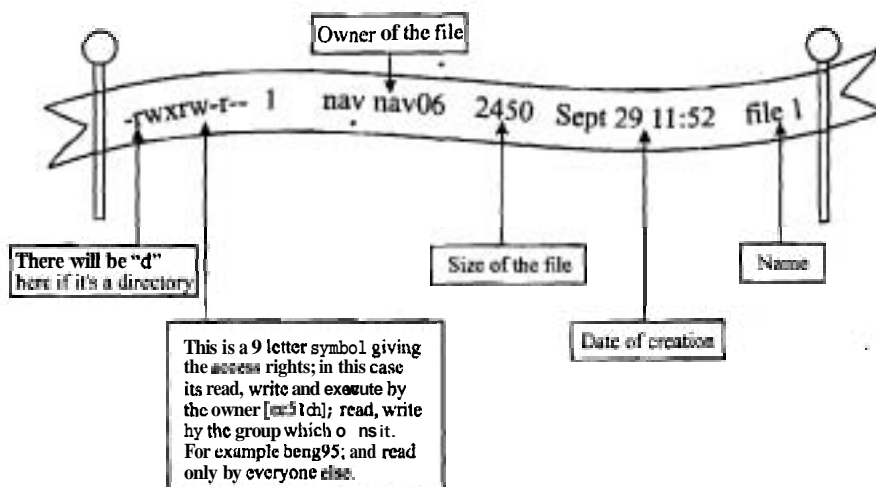| Man command options	Meaming
– t	Phototypeset the section using troff(1).
– n	Print the section on the standard output using nroff(1).
– k	Display the output on a Tektronix 4014 terminal using
– e	Appended or prefixed to any of the above causes the manual section to be

For example,

? To find out more about the wc (word count) command

% man wc

File access rights

When you give ls -l command you get lots of details about the contents of your directory, similar to the example given below:

-rwxrw-r-- 1 nav nav06 2450 Sept29 11:52 file1

-rwxrw-r-- 1 ee51ab beng95 2450 Sept29 11:52 file1

Owner of the file

-rwxrw-r-- 1 nav nav06 2450 Sept 29 11:52 file 1

There will be "d" here if it's a directory

Size of the file

Name

Date of creation

This is a 9 letter symbol giving the access rights; in this case its read, write and execute by the owner [ee51ch]; read, write by the group which o ns it. For example beng95; and read only by everyone else.

In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x, -, and, occasionally, s or S. If d is present, it will be at the **left** hand end of the string, and indicates a directory: otherwise - will be the starting symbol of the string. The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3 each.

- The **left** group of 3 gives the file permissions for the user that owns the file (or directory)
- The middle group gives the permissions for the group of people to whom the file (or directory) belongs
- The rightmost group gives the permissions for all others.

The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory. Access rights for files are given below:

- r (or -), indicates read permission (or otherwise), that is, the presence or absence of permission to read and copy the file
- w (or -), indicates write permission (or otherwise), that is, the permission (or otherwise) to change a file
- x (or -), indicates execution permission (or otherwise), that is, the permission to execute a file, where appropriate.

Access rights for directories

- r allows users to list files in the directory;
- w means that users may delete files from the directory or move files into it;
- x means the right to access files in the directory for execution.

Lets take a few examples to understand the access rights:

-rwxrwxrwx   a file that everyone can read, write and execute.

-rw-------   a file that only the owner can read and write - no-one else can read.

chmod

To change the access right of a file use chmod command that will change the mode of the file. Only the owner of a file can use chmod to change the permissions of a file. The options of chmod are as follows':

Symbol	Meaning
u	user
g	group
o	other
a	all
r	read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

For example,

❓ To remove, read, write, and execute permissions on the file **biglist** for the group and others
    % chmod go-rwx **biglist**.
❓ To give read and write pennissions on the file **biglist** to all,
    % chmod a+rw **biglist**.   .

We can make some shortcut options for changing the mode. It can be constructed by ORing together some combination of numbers as given below:

Combination	Meaning
04000	set user ID on execution
02000	set group ID on execution
01000	save text image after execution
00400	read by owner
00200	write by owner
00100	execute (search on directory) by owner
00070	read, write, execute (search) by group
00007	read, write, execute (search) by others

Command for Processes and Jobs

A process is an executing program identified by a unique PID (process identifier).

? To see information about your processes with their associated PID and status,

⚷ % ps

A process may be in the foreground, in the background, or be suspended. In general the shell does not return the Unix prompt until the current process has finished executing.

Some processes take a long time to run and hold up the terminal. Using the facility of multitasking the Unix prompt is returned immediately when a long process is sent to the background, and other tasks can be carried out while the original process continues executing. To make a process to run in the background, use and symbol at the end of the command line. For example, the command sleep command waits a given number of seconds before continuing, that is why % sleep 100 will hang up you for100 seconds. To utilize the time properly you can use **&** and send sleep in background.

? To run **sleep** in the background, **type**

% **sleep** Y0 **&**

The **&** runs the job in the background and returns the prompt. When sleep command will finish it will return the job number and PID (process ID). Background execution is useful for jobs which will take a long time to complete. You can also suspend the process running in the foreground by holding down the [control] key and typing [z] (written as ^Z).

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, type % jobs an example of a job list could be as given below with the job number in [] brackets:

[11] Suspended sleep 100

[12] Running netscape

[33] Running nedit

To restart a suspended processes write % fg **%jobnumber.** For example,

? To restart sleep 100

⚷ % fg %11

It is sometimes necessary to kill a process (for example, when an executing program is in an infinite loop) .To kill a job running in the foreground, type ^C (control c). To kill a suspended or background process, type % kill **%jobnumber.** Otherwise, processes can be killed by finding their process numbers (PIDs) and using kill **PID_number,** If a process refuses to be killed, uses the -9 option, i.e. write

% kill -9 **PID_number**

Miscellaneous

quota

? To check your current quota of amount of disk space on the file system and how much of it you have used

    % quota **-v**

Disk free (df)

The df command reports on the space left on the file system. For example,

? To find out how much space is left on the **fileserver,** type

🔑 % df .

Disk Utilized (du)

The du command outputs the number of kilobytes used by each subdirectory. Useful if you have gone over you quota and you want:

? To find out which directory has the most files. In your home-directory,

    % du

Compress

This reduces the size of a file, thus freeing valuable disk space. For example, type % ls -l science.txt and note the size of the file. Then to compress science.txt, type % compress science.txt. This will compress the file and place it in a file called science.txt.Z. To see the change in size, type ls -l again. To uncompress the file, use the uncompress command. % uncompress science.txt.Z

gzip

This also compresses a file, and is more efficient than compress. For example,

? To zip science.txt, type

🔑 % gzip science.txt

This will zip the file and place it in a file called science.txt.gz

? To unzip the file, use the gunzip command.

🔑 % gunzip science.txt.gz

File

File classifies the named files according to the type of data they contain, for example, ASCII (text), pictures, compressed data, etc.

? To report on all files in your home directory, type

🔑 % file *

history

The C shell keeps an ordered list of all the commands that you have catered. Each command is given a number according to the order it was entered. % history command shows the command history list. If you are using the C shell, you can use the exclamation character (!) to recall commands easily.

Command	Meaning
% !!	Recall last command
% !-3	Recall third most recent command
% !5	Recall 5th command in list
% !grep	Recall last command starting with grep

1.4   SUMMARY

In this section we studied that Unix that is know as one of the most versatile, flexible and powerful operating systems in the computer world, Unix was created at Bell Labs in 1970 written in the C programming language. We studied that the command, files and directories are the basic **components of** Unix. We have also studied the Unix architecture, which **is divided into three layers Application Program, System calls and Kernel. This section also covered the detailed description of Unix command those are** used to manipulate **files,** data and environment. The next section of this manual covers the basic introduction to the C. language.

1.5   FURTHER READINGS

1)   **http://www.programmersheaven.com**

2)   http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html

3)   http:Nrcsg-gsir.imsb-dsgi.nrc-cnrc.gc.caldocuments/bagiclnode105.html.

4)   **Sumitabha Das,** *Unix Concepts and applications* Tata **Mcgraw** Hill, 2003

5)   www.rice.edu/Computer/Documents/Unix/unix1.pdf