BSM 308-System Programming

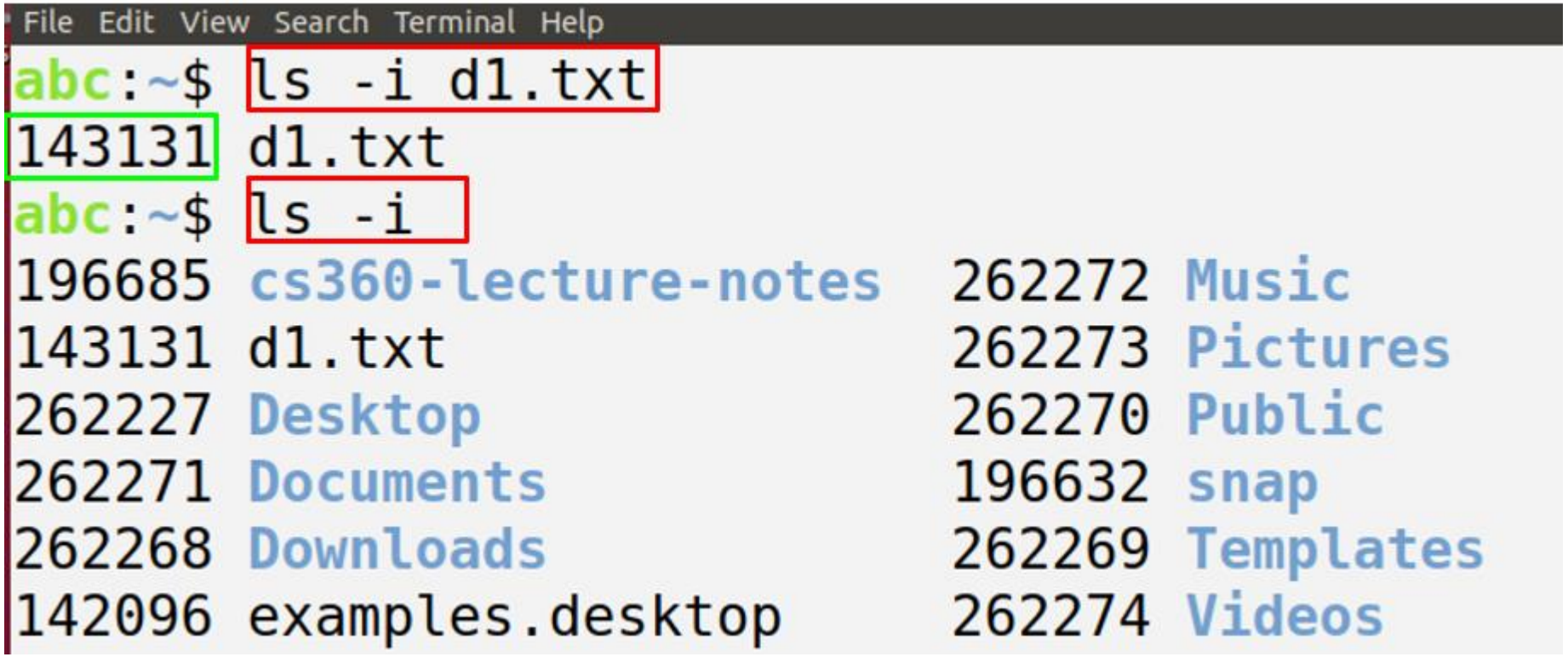
Sakarya University-Computer Engineering

Files

- Files store information in secondary storage.
- This means that the information should be available even if the computer on which it is stored is turned off.
- This is in contrast to primary storage (RAM, etc.), which only works when the computer is on and is lost forever when the computer is turned off.
- When you create a file in Unix, there are many things that happen.
- In this lesson, we will focus on the three components of a file in Unix:
  - Bytes of the file itself.
  - Metadata of the file.
  - Links to file relative to a directory.

Files

- ls -la list long format including hidden files
- ls -i inode directory number of the list file



Files

We created a file called f1.txt and this places three things on disk:

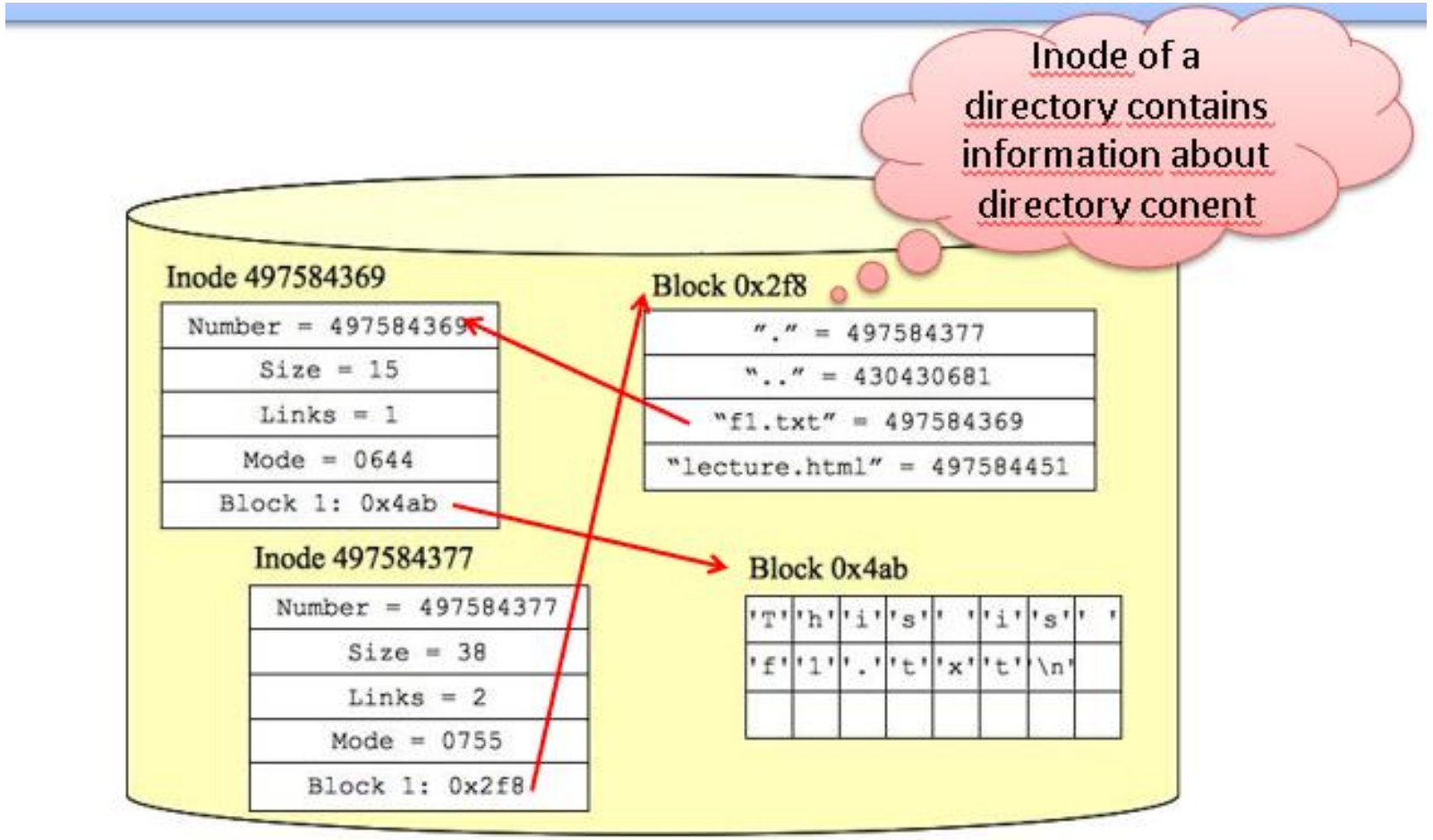
1-) A directory entry for "f1.txt" in the current directory. This associates the name "f1.txt" with an " inode " whose number is 397648. An Inode with number 397648.

2-) This contains metadata or information about the file. Typical metadata is size, owner, links, protection mode, last access and modification time, etc. ' is . This is stored in its own location on disk -- the operating system knows how to find it by inode number.

3-) Real bytes, "This is f1.txt". These are placed in a disk block and the inode has information on how to find the disk block. In this example, the file is located within a disk block and the inode will know how to find it.

Files

```
UNIX> echo "This is f1.txt" > f1.txt
UNIX> ls -lai
total 20
497584377 drwxr-xr-x  2 plank guest   38 Feb  3 13:54 .
430430681 drwxr-xr-x 51 plank guest 4096 Feb  3  2014 ..
497584369 -rw-r--r--  1 plank loci    15 Feb  3 13:54 f1.txt
497584451 -rw-r--r--  1 plank guest 9896 Feb  3 13:44 lecture.html
UNIX>
```



Files

- You can see how it shows the 0x4ab block, which contains the inode for f1.txt (397648) and the bytes of the file (you don't have access to this information unless you're the operating system – random 0x4ab number).
- We added information to the directory, which is itself a file on disk, and to the inode of this file. See how everything connects?
- You'll also notice that I didn't put a null character at the end of the string in the disk block. This is because there are no null characters -- this only exists when you use a string within a C program.
- There are no blank characters when you write to disk. When you give the -i flag to ls it will tell you the inode number like in the example above .

Files

- In Unix , the way we name a file is by adding a "link" to the inode .
- in the inode that points to the file. Maps to inode number. We can have multiple ports to a file.
- Let's say we are in a new directory and create the file f1 to contain the bytes "This f1\n".
- Also, assume the inode number of this file is 34778. Now we do the following:
- This says to create another link to file f1 and name it "f2".

UNIX > **ln f1 f2**

- This link is really an entry in the directory that maps "f2" to inode 34778 . What we have now are two pointers to the same metadata and the same bytes on disk.

Creating a link

There may be more than one link pointing to a file

```
File  Edit  View  Search  Terminal  Help
abc:test$ echo "This is the content of f1.txt" >f1.txt
abc:test$ ls -li f1.txt
263287 -rw-r--r-- 1 abc abc 30 Apr  5 19:56 f1.txt
```

ln creates a link between files.

```
abc:test$ ln f1.txt f2.txt
abc:test$ ls -li f1.txt f2.txt
263287 -rw-r--r-- 2 abc abc 30 Apr  5 19:56 f1.txt
263287 -rw-r--r-- 2 abc abc 30 Apr  5 19:56 f2.txt
```

f1.txt and f2.txt points to the same file. They have the same inode.

Number of links pointing to the same file.

Creating a link

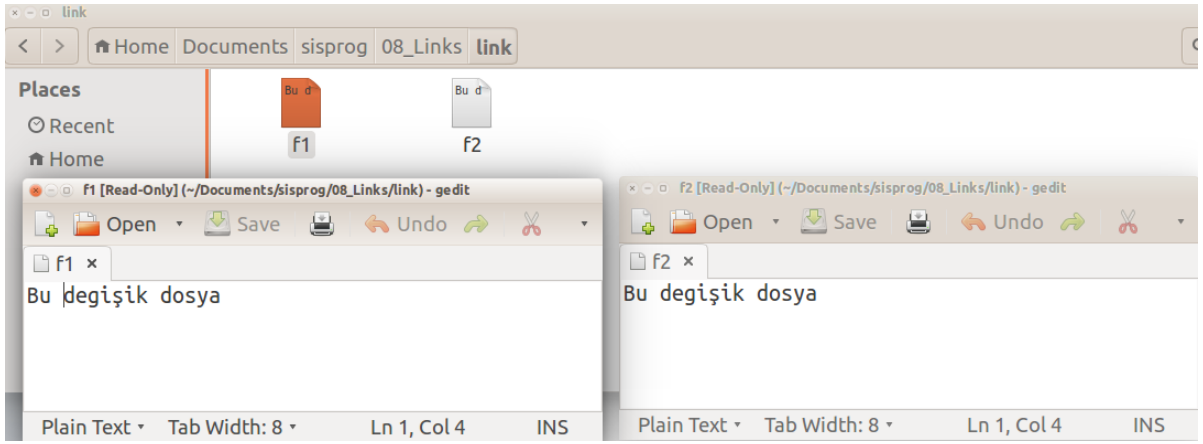
There may be more than one link pointing to a file

Contents of the file are the same

```
abc:test$ ls -li f1.txt f2.txt
263287 -rw-r--r-- 2 abc abc 30 Ap
263287 -rw-r--r-- 2 abc abc 30 Ap
abc:test$ cat f1.txt f2.txt
This is the content of f1.txt
This is the content of f1.txt
abc:test$
```

File content

```
File Edit View Search Terminal Help
link: cat f1
Bu degişik dosya
link: cat f2
Bu degişik dosya
link:
```




```
File Edit View Search Terminal Help
link: ls -li f1 f2
1713621 -rw-r--r-- 2 root root 18 Mar  7 21:13 f1
1713621 -rw-r--r-- 2 root root 18 Mar  7 21:13 f2
link:
```

Both have the same edit time because the edit time is stored as part of the file.

Changes on links

If we change the protection mode of one file, it changes for the other.

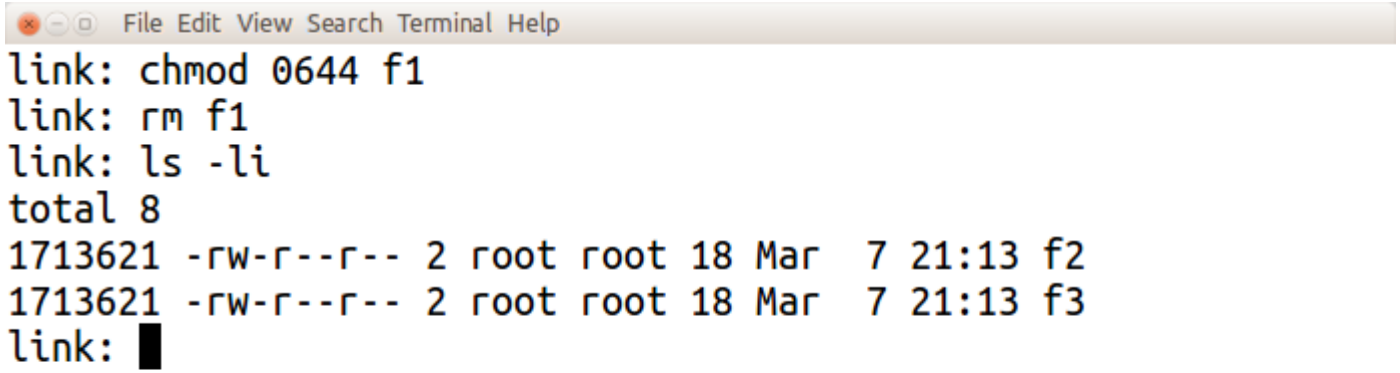
```
File Edit View Search Terminal Help
link: chmod 0400 f1
link: ls -li
total 8
1713621 -r-------- 2 root root 18 Mar  7 21:13 f1
1713621 -r-------- 2 root root 18 Mar  7 21:13 f2
link:
```

- The number of links to the file can be increased.

```
File Edit View Search Terminal Help
link: ln f2 f3
link: ls -li
total 12
1713621 -r-------- 3 root root 18 Mar  7 21:13 f1
1713621 -r-------- 3 root root 18 Mar  7 21:13 f2
1713621 -r-------- 3 root root 18 Mar  7 21:13 f3
link:
```

Removing link

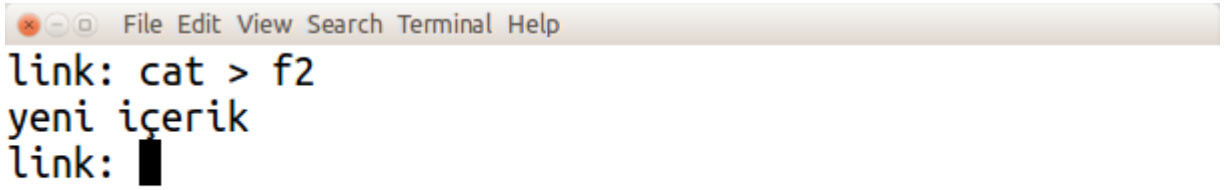
Link can be removed using **rm**



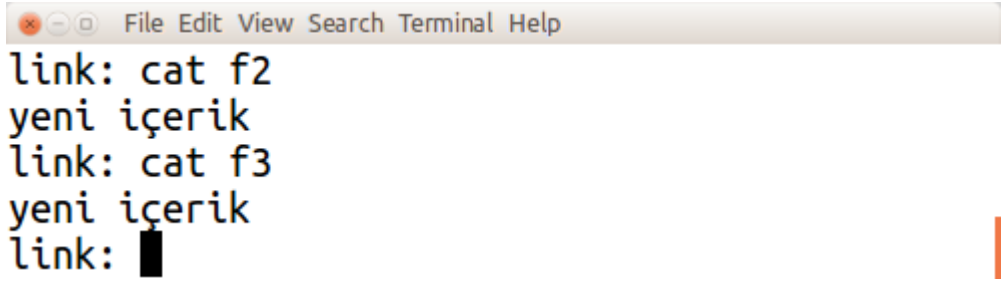
- The file is not deleted until the last link is deleted.

Overwrite

Creating an f2 file with Cat and adding new content does not delete the existing file.

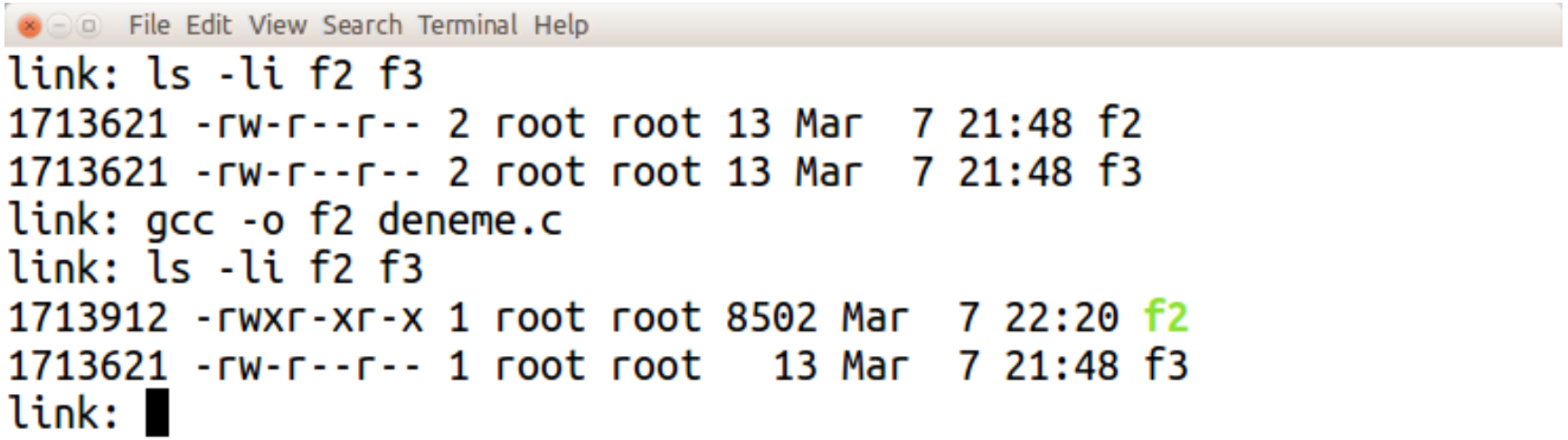


Links f2 and f3 continue to show the same file.



Overwrite

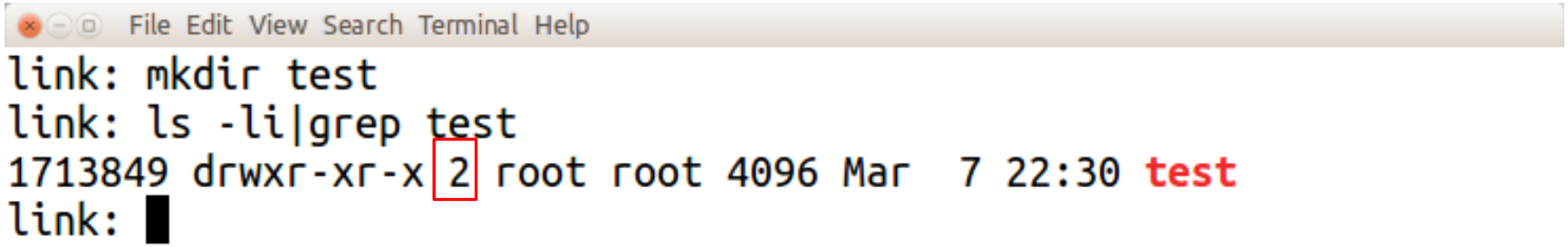
When we compile a file named f2 with the C compiler, we first rm f2 and create a new executable named f2.

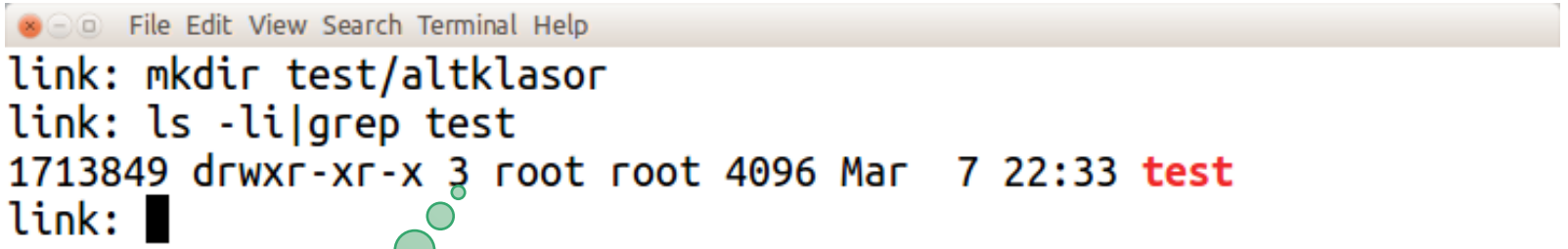


The number of links

Each newly created directory contains “.” and “..” subdirectories.

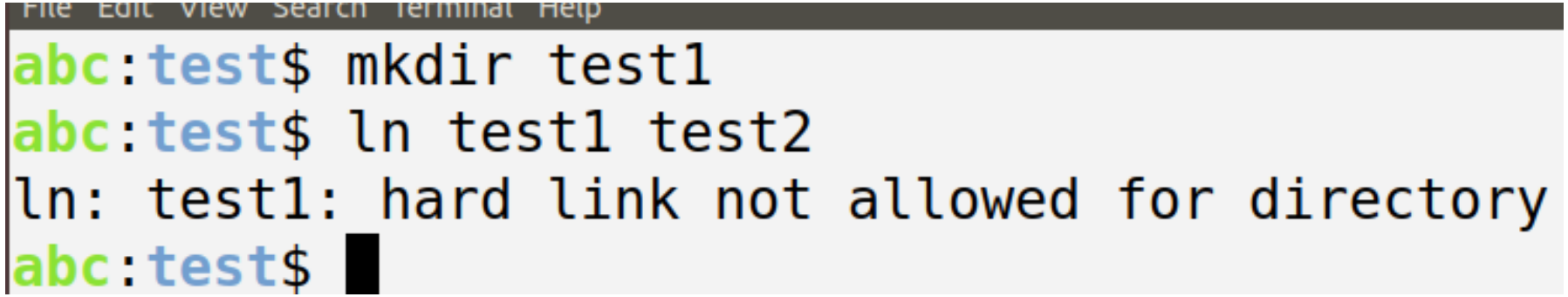
The first one points to the same directory and the other one points tyo the parent directory.





We created a new subfolder and the number of links increased to 3.

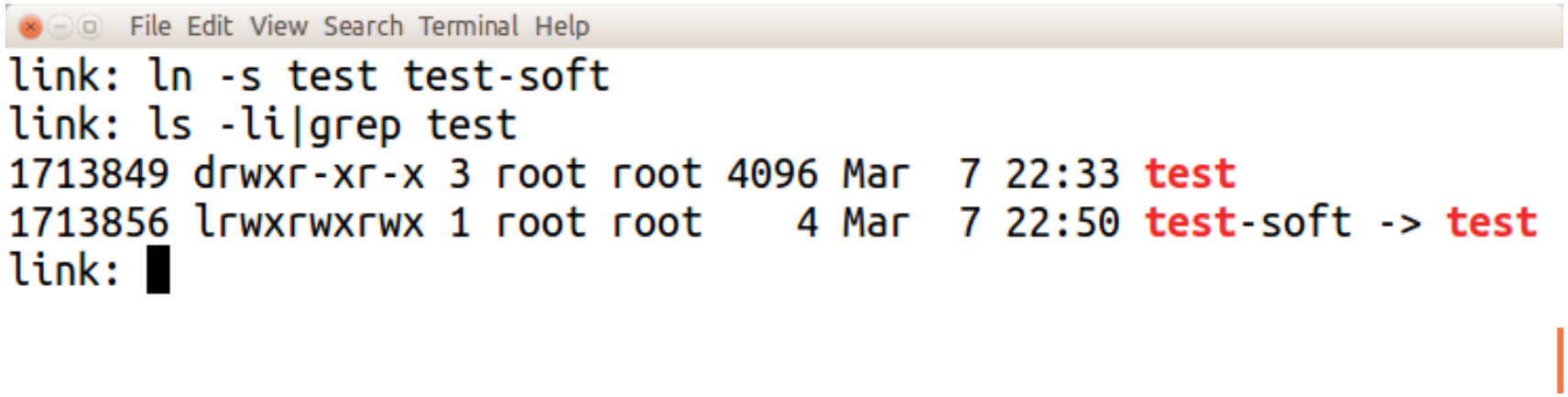
Hard link for a directoriy is not alllowed



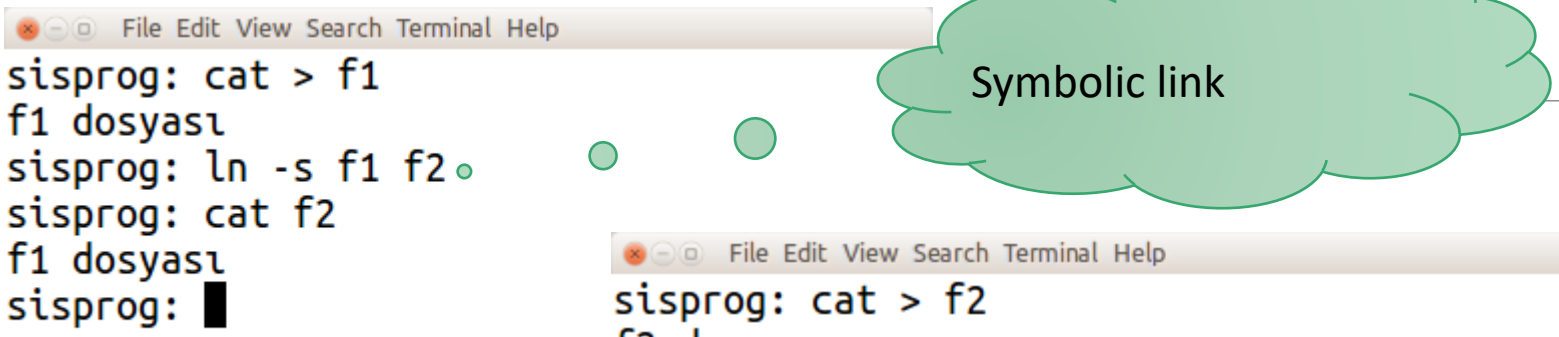
Soft link (Symbolic link)

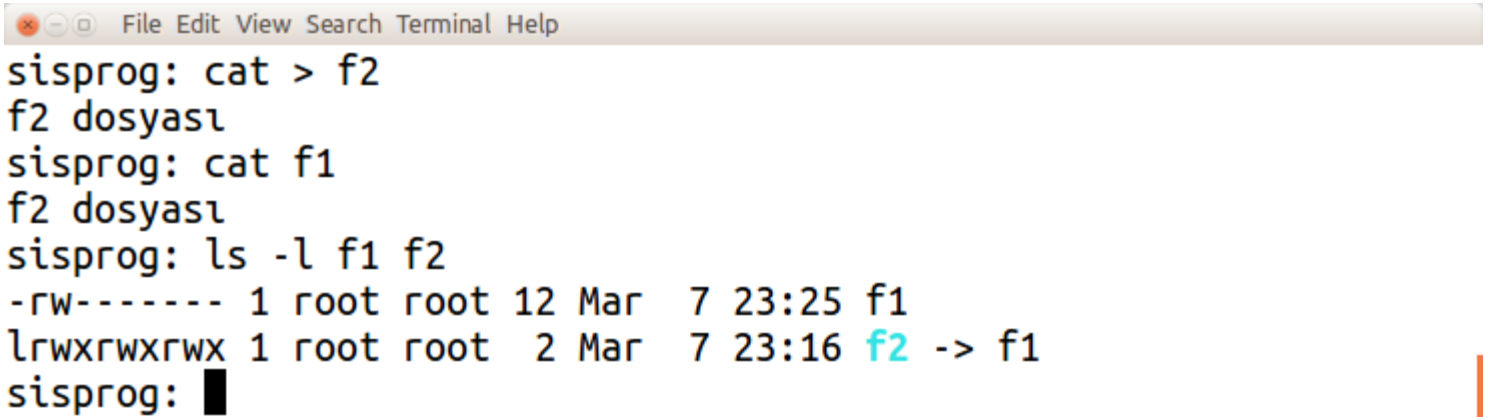
- Hard links : Refer to the specific location of physical data.

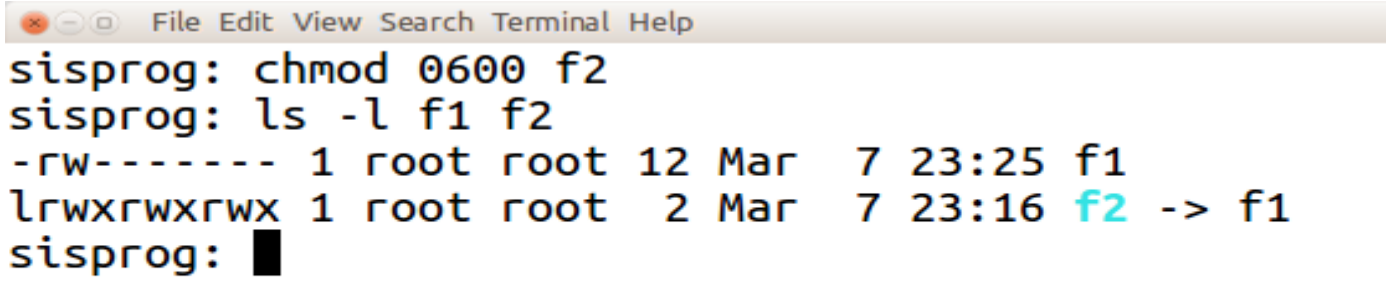
- Symbolic links: Refer to a symbolic path indicating the abstract location of another file
- Soft link can be created with ln -s.
- Soft links point to the file without modifying the inode.



Soft link (Symbolic link)

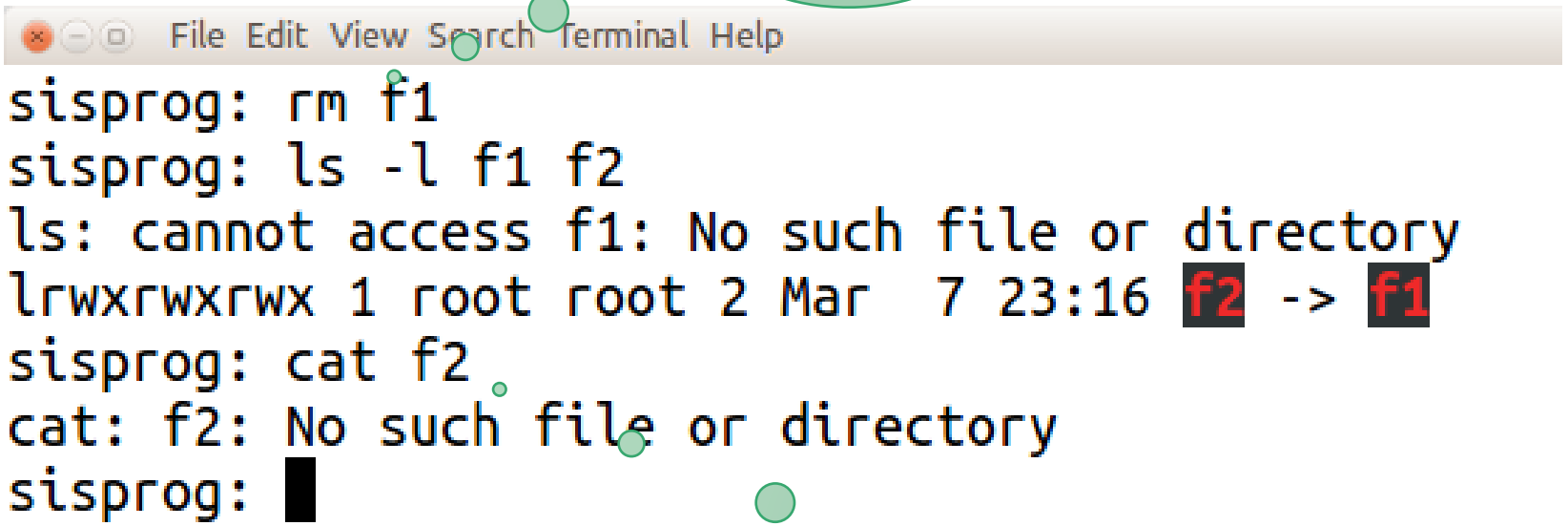






Soft link

Hardlink is removed



An error occurs when we try to access it via the symbolic link.

inode

- Hard links have the same inode number as the source file.
- Soft links have a different inode number than the source file.
- When the last hardlink is deleted, the file is deleted even if the soft links are not deleted.
- A hardlink cannot be created from one file system to another.

inode

Making shortcut:

Documents: ln -s /home/bilg/Documents/deneme.txt /home/bilg/Desktop/kısayoldeneme

Deleting the file using its the inode number

find -inum 1969436 **-delete**

find -inum

```
abc:test$ ln f3.txt f4.txt
abc:test$ ls -li
total 16
263399 -rw-r--r-- 4 abc abc 10 Apr  7
263399 -rw-r--r-- 4 abc abc 10 Apr  7
263399 -rw-r--r-- 4 abc abc 10 Apr  7
263399 -rw-r--r-- 4 abc abc 10 Apr  7
abc:test$ cat f4.txt
zxcvbnm
abc:test$ find -inum 263399
./f2.txt
./f4.txt
./f3.txt
./f1.txt
```

Finds all files with the same inode

Finds and deletes all files with the same inode

```
abc:test$ find -inum 263399 -delete
abc:test$ ls
abc:test$
```

Shell Redirection

- There are many different shells that people use under Unix.
- Bourne Shell with some references to the C shell . The " bash " shell is derived from the Bourne Shell, so your Bourne Shell scripts will run on bash .
- What we are interested in here are routing primitives.
- Most of these are simple and almost the same in all shells.

The basic redirection functions are used similarly on Bourne shell and C shell.

> d : Writes standard output to the file d

>> d : Appends standard output to the file d.

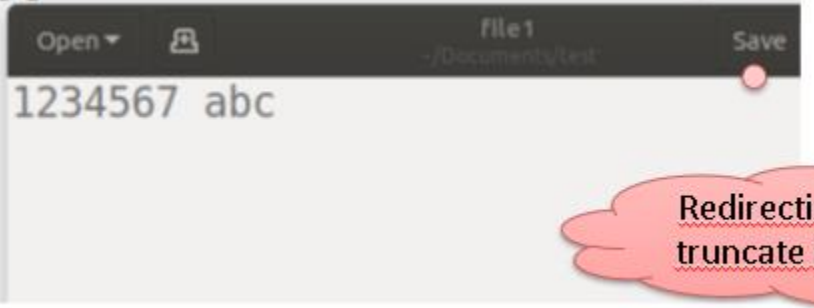
< d : Takes standard input from the file d.

echo prints the given string to stdout.

```
File  Edit  View  Search  Terminal  Help
abc:test$ echo "System programming"
System programming
abc:test$ echo "System programming" > file1
abc:test$ ls -l
total 4
-rw-r--r-- 1 abc abc 19 Apr  5 22:47 file1
```

Stdout is redirected to file

```
File  Edit  View  Search  Terminal  Help
abc:test$ cat file1
System programming
abc:test$ echo "1234567 abc" > file1
abc:test$ cat file1
1234567 abc
abc:test$
```

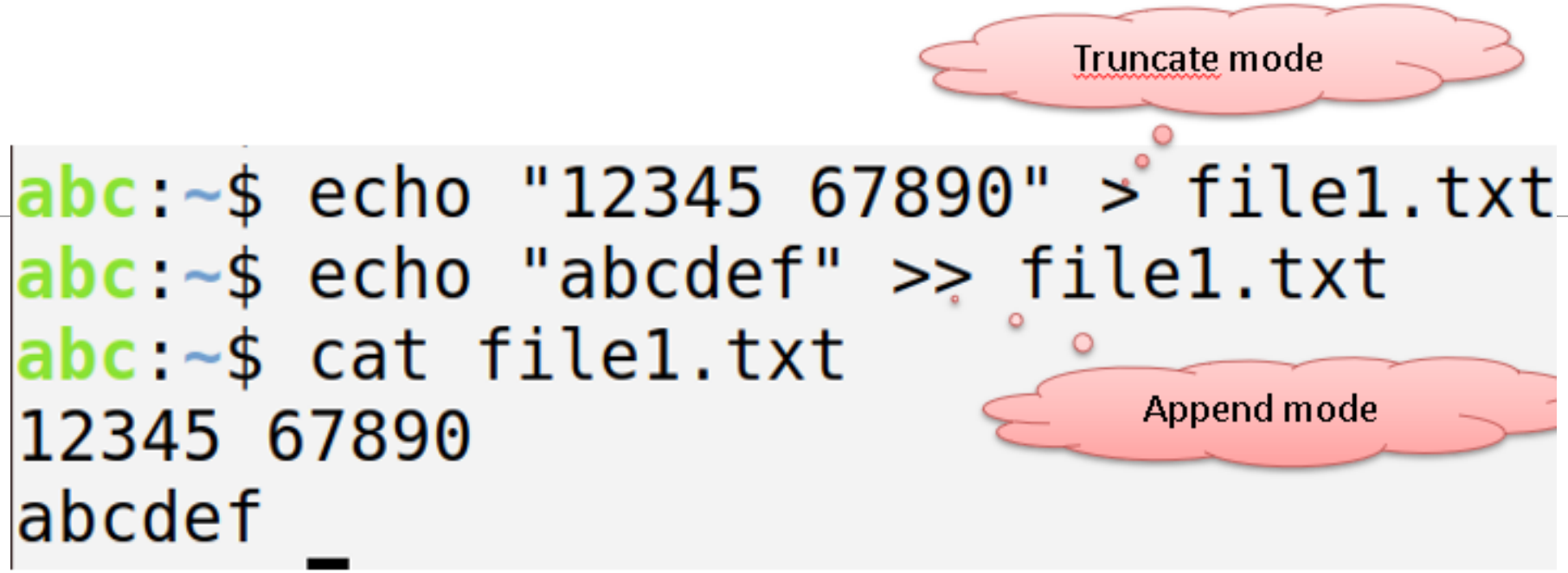


Redirection to file1 truncate its content

```
abc:test$ ./prog1
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
abc:test$
```

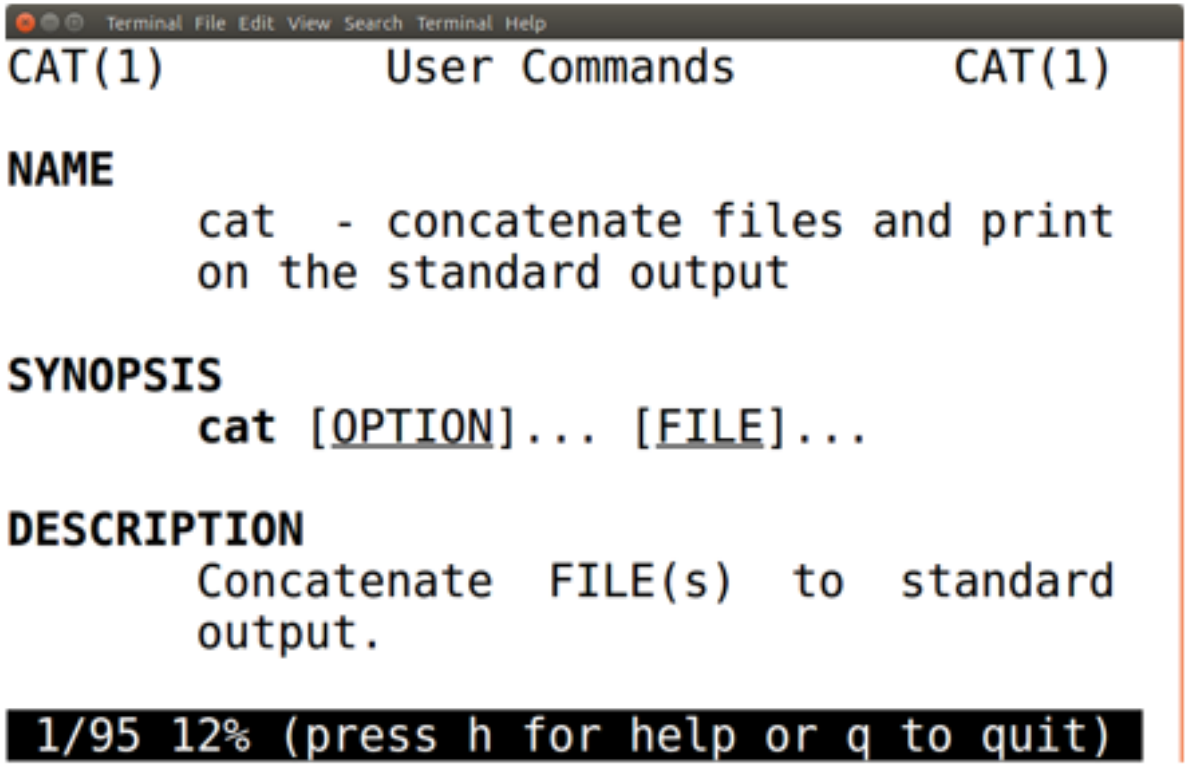
```
i=0
i=7
i=8
i=9
abc:test$ ./prog1 > file1
abc:test$ cat file1
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
```

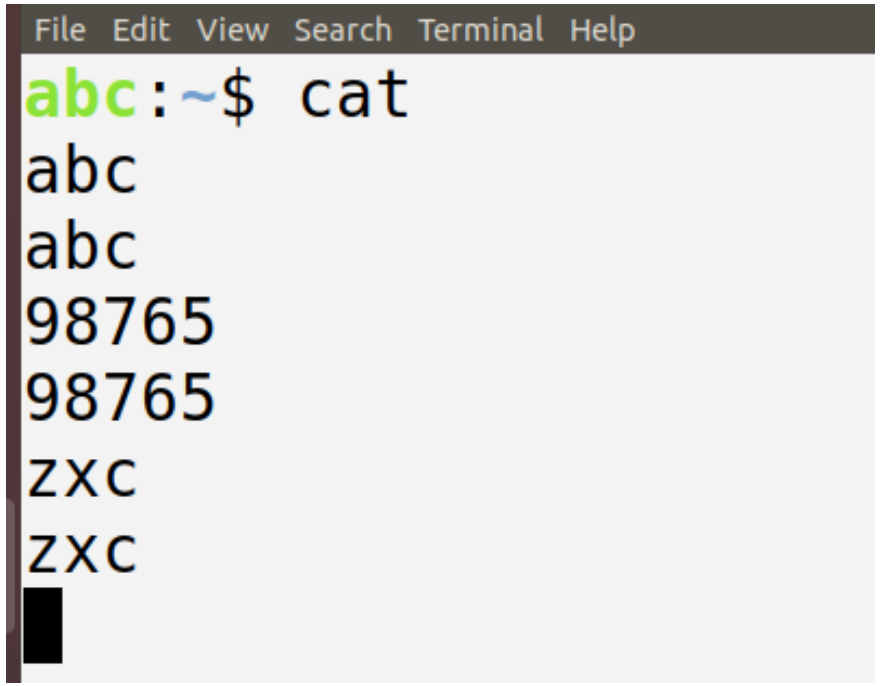
Output of the program is redicted to file1



- While > truncates the content >> works in append mode

cat is one of the frequently used Shell commands.





If we just write cat and press enter, it writes the entries entered on the screen again. In other words it takes ***stdin*** and directs it to ***stdout***.

```
File  Edit  View  Search  Terminal  Help
abc:~$ cat file1.txt
12345 67890
abcdef
abc:~$ echo "zyxw" >file2.txt
abc:~$ cat file1.txt file2.txt
12345 67890
abcdef
zyxw
abc:~$ cat file1.txt file2.txt >file3.txt
abc:~$ cat file3.txt
12345 67890
abcdef
zyxw
abc:~$
```

Cat prints the contents of two files

We can merge the files into one file

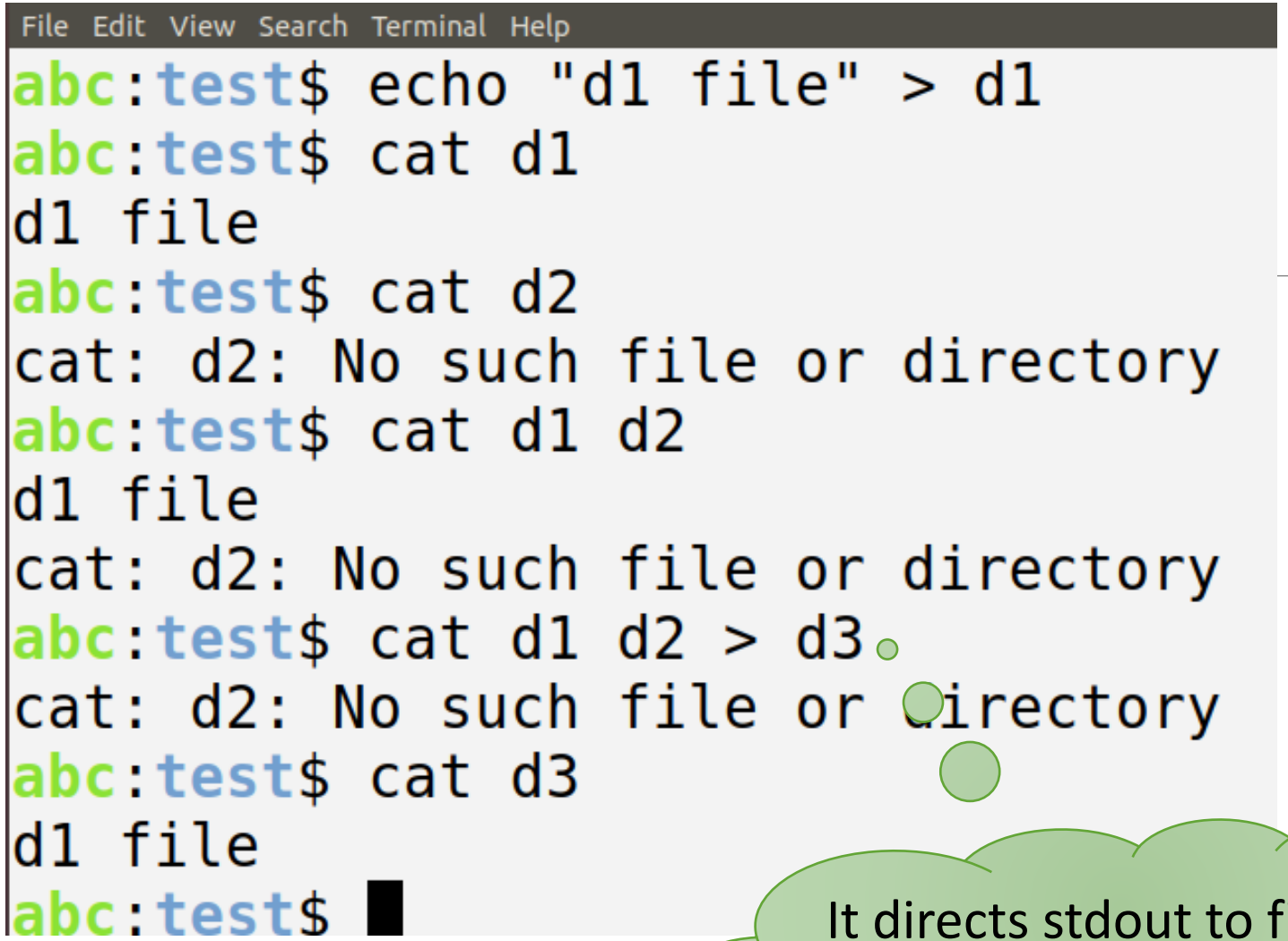
```
abc:~$ cat <file1.txt
12345 67890
abcdef
abc:~$
```

```
File  Edit  View  Search  Terminal  Help
abc:test$ ./prog2
abcde
Entered string: abcde
abc:test$
```

```
~/Documents/test
#include <stdio.h>
#include <stdlib.h>
int main(){
char str[100];
scanf("%s",str);
printf("Entered string: %s\n",str);
return 0;
}
```

```
File  Edit  View  Search  Terminal  Help
abc:test$ echo "abcde" >file1.txt
abc:test$ cat file1.txt
abcde
abc:test$ ./prog2 <file1.txt
Entered string: abcde
abc:test$
```

Redirection to standart input.



It directs stdout to file. Error message printed using stderr stream.

Shell redirection

```
abc:test$ cat d1 d2 1> d3
cat: d2: No such file or directory
abc:test$
```

It directs stdout (1) to file. Error message printed using stderr (2) stream.

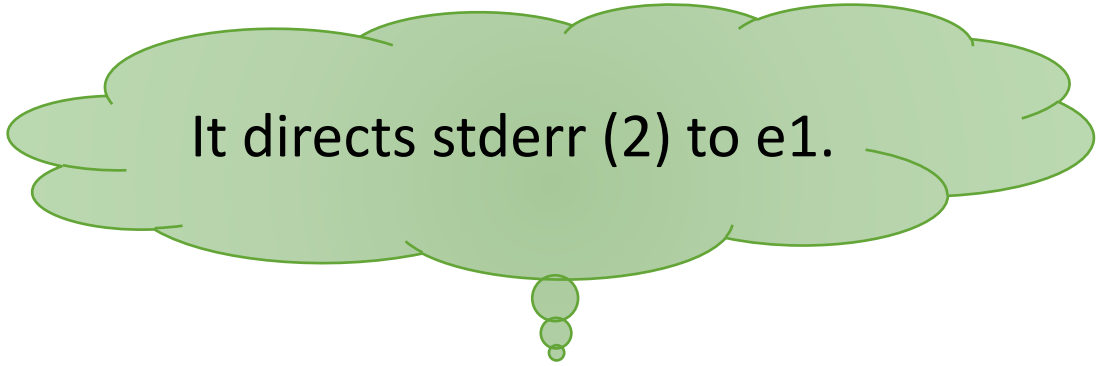
File descripters:

0: stdin

1: stdout

2: stderr

Shell redirection




```
abc:test$ cat d1 d2 1>d3 2>e1
abc:test$ cat d3
d1 file
abc:test$ cat e1
cat: d2: No such file or directory
```

Shell redirection

```
abc:test$ cat d1 d2 >& d3
abc:test$ cat d3
d1 file
cat: d2: No such file or directory
abc:test$
```

Stdout and stderr merged together and directed to d3

```
abc:test$ cat d1 d2 >d3  2>&1
abc:test$ cat d3
d1 file
cat: d2: No such file or directory
abc:test$
```

Alternative to above

File Edit View Search Tools Documents Help

Open Save Undo

*badbadcode.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
  char *s = "fd=3 dosyası\n";
  int i;
  //int fd;

  i = write(3, s, strlen(s));
  printf("%d\n", i);
  if (i < 0) perror("write");
}
```

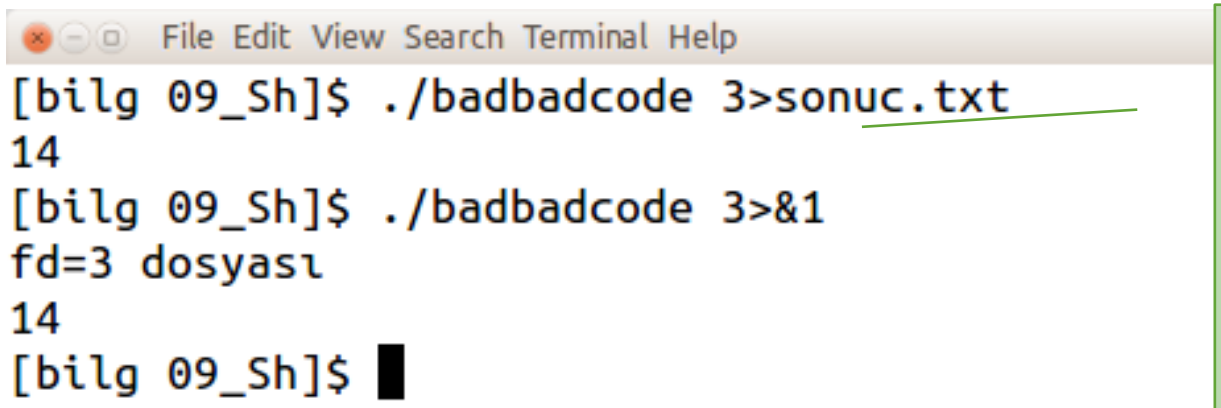
C Tab Width: 8 Ln 17, Col 1 INS

s character array is written to a file described by 3 file descriptor.
But this file should be opened before writing something.

File Edit View Search Terminal Help

```
[bilg 09_Sh]$ ./badbadcode
-1
write: Bad file descriptor
[bilg 09_Sh]$
```

Shell redirection



If the file descriptor specified with 3 in the code is directed to a file such as sonuc.txt, the program will not generate an error. Alternatively fd=3 can be redirected to 1 and the message is printed on the screen.

Shell Redirection

- For example, assume the file f1 contains the bytes `` This is f1''. The directions below should not confuse you. In any case, ask yourself what the output of the command should be:

```
UNIX> cat f1
This is f1
UNIX> cat < f1
This is f1
UNIX> < f1 cat       You can put the redirection anywhere in the command line.
This is f1
UNIX> < cat f1       This is the same as f1 < cat - it can't find the file "cat".
cat: No such file or directory.
UNIX> cat f1 > f2
UNIX> cat f2
This is f1
UNIX> cat f1 >> f2
UNIX> cat f2
This is f1
This is f1
UNIX> > f2 < f1 cat      This is the same as cat < f1 > f2
UNIX> cat f2
This is f1
UNIX>
```

Shell Redirection

- Now let's consider x>y again. If you specify y as &y, it ensures that the x file descriptor in the program is the same as the y file descriptor.
- Another way to say this is: "Wherever file descriptor y is now going, so is file descriptor x now, and they're the same."

```
UNIX> cat f1 f3 > f2 2>&1
UNIX> cat f2
This is f1
cat: cannot open f3
```

Shell Redirection

- What's happening? First, you route the standard output to f2. This means file descriptor 1 will go to f2.
- Then the 2>&1 part tells you to make file descriptor 2 the same as file descriptor 1. This means the standard error will also go to f2.
- Again, these are processed by the shell from left to right. Let's say you reverse the order of the expressions:

```
UNIX> cat f1 f3 2>&1 > f2
cat: cannot open f3
UNIX> cat f2
This is f1
```

Shell Redirection

- Now, the 2>&1 part tells it to make file descriptor 2 the same as file descriptor 1, which goes to the screen as soon as the shell sees this command.
- It then redirects file descriptor 1 to f2. So standard error goes to the screen and standard output goes to f2.

```
UNIX> cat f1 f3 >f2 2>&1 1>f5
UNIX> cat f2
cat: cannot open f3
UNIX> cat f5
This is f1
```

Assignment

When the program is run, perform the following operations using redirection commands.

*** first print the values of the array to the screen**

*** the values of the index will be written to the screen and the address values will be written to the file you will give as a parameter**

*** both the values of the array and the address values will be printed on the screen**

File Edit View Search Tools Documents Help

Open Save

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
char str[65];
int i;
//int * dizi=(int *)malloc(5*sizeof(int));
int dizi[5];

for (int k=0;k<5;k++)
  dizi[k]=rand()%100;

for (int k=0;k<5;k++){
  sprintf(str,"&dizi[%d]=0x%lx\n",
  k,(long unsigned int)&dizi[k]);
  i = write(3, str, strlen(str));
  printf("dizi[%d]=%d\n",k,dizi[k]);
}
}
```

C Tab Width: 2 Ln 20, Col 2 INS

```
Terminal File Edit View Search Terminal Help
bilg:8_Sh$ ./cikti
dizi[0]=83
dizi[1]=86
dizi[2]=77
dizi[3]=15
dizi[4]=93
bilg:8_Sh$ ./cikti 3>dosya
dizi[0]=83
dizi[1]=86
dizi[2]=77
dizi[3]=15
dizi[4]=93
bilg:8_Sh$ cat dosya
&dizi[0]=0x7ffef07a8300
&dizi[1]=0x7ffef07a8304
&dizi[2]=0x7ffef07a8308
&dizi[3]=0x7ffef07a830c
&dizi[4]=0x7ffef07a8310
bilg:8_Sh$
```

```
File Edit View Search Tools Documents Help
Open  Save
&dizi[0]=0x7ffef07a8300
&dizi[1]=0x7ffef07a8304
&dizi[2]=0x7ffef07a8308
&dizi[3]=0x7ffef07a830c
&dizi[4]=0x7ffef07a8310
```

```
Terminal File Edit View Search Terminal Help
bilg:8_Sh$ ./cikti 3>&1
&dizi[0]=0x7ffeaeedc250
dizi[0]=83
&dizi[1]=0x7ffeaeedc254
dizi[1]=86
&dizi[2]=0x7ffeaeedc258
dizi[2]=77
&dizi[3]=0x7ffeaeedc25c
dizi[3]=15
&dizi[4]=0x7ffeaeedc260
dizi[4]=93
bilg:8_Sh$
```