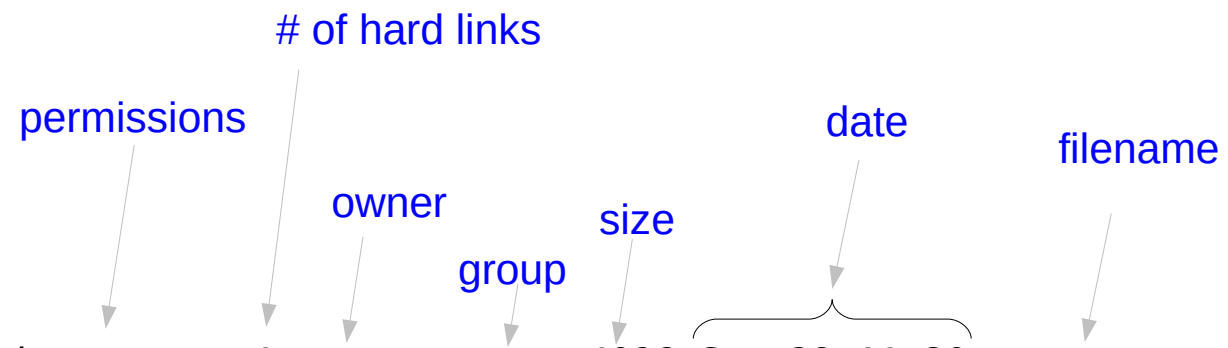
File System (1A)

Young Won Lim
10/24/25

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

ls




```
drwxrwxr-x 4 young young  4096 Sep 29 11:20 .
drwxrwxr-x 7 young young  4096 Sep 12 15:20 ..
-rw-rw-r-- 2 young young    20 Sep 28 06:32 a.txt.hard.link
lrwxrwxrwx 1 young young    20 Sep 28 08:27 a.txt.soft.link -> d1/d1_1/d1_1_1/a.txt
```

chmod chown chgrp

Unix File Types

File types

```
drwxrwxr-x 4 young young  4096 Sep 29 11:20 .
drwxrwxr-x 7 young young  4096 Sep 12 15:20 ..
-rw-rw-r-- 2 young young    20 Sep 28 06:32 a.txt.hard.link
lrwxrwxrwx 1 young young    20 Sep 28 08:27 a.txt.soft.link -> d1/d1_1/d1_1_1/a.txt
```

```
-rw-r--r-- ... /etc/passwd    Regular File
drwxr-xr-x ... /              Directory File
lrwxrwxrwx ... termc-> ..     Symbolic Link
prw-rw---- ... mypipe         Named Pipe
srwxrwxrwx ... /tmp/XO        Socket
crw------- ... /dev/null      Character Device File
brw-rw---- ... /dev/sda       Block Device File
```

ls

chmod

```
drwxrwxr-x 4 young young  4096 Sep 29 11:20 .
drwxrwxr-x 7 young young  4096 Sep 12 15:20 ..
-rw-rw-r-- 2 young young    20 Sep 28 06:32 a.txt.hard.link
lrwxrwxrwx 1 young young    20 Sep 28 08:27 a.txt.soft.link -> d1/d1_1/d1_1_1/a.txt
drwxrwxr-x 3 young young  4096 Sep 28 06:28 d1
-rw-rw-r-- 1 young young   942 Sep 14 17:15 gcc.txt
-rw-rw-r-- 1 young young   594 Sep 12 15:34 gcc.txt~
drwxrwxr-x 3 young young  4096 Sep 13 10:39 one
lrwxrwxrwx 1 young young    15 Sep 28 06:32 softln -> d1/d1_1/d1_1_1/
-rw-r--r-- 1 young young 24806 Sep 28 06:40 SysP.1.A.FIle.20120928.odp
-rw-rw---- 1 young young    86 Sep 13 09:33 t.c
-rwxrwxr-x 1 young young  8373 Sep 13 09:40 t.exe
-rw-rw-r-- 1 young young  1496 Sep 13 09:39 t.o
-rw-rw-r-- 1 young young   323 Sep 13 09:30 tt.c
lrwxrwxrwx 1 young young    47 Sep 13 10:44 two-2 ->
/home/young/Documents/Work/Work.SysProg/one/two
```

chmod chown chgrp

Softlink

```
lrwxrwxrwx 1 young young    20 Sep 28 08:27 a.txt.soft.link -> d1/d1_1/d1_1_1/a.txt
lrwxrwxrwx 1 young young    15 Sep 28 06:32 softln -> d1/d1_1/d1_1_1/
lrwxrwxrwx 1 young young    47 Sep 13 10:44 two-2 ->
/home/young/Documents/Work/Work.SysProg/one/two
```

Hardlink

```
-rw-rw-r-- 2 young young    20 Sep 28 06:32 a.txt.hard.link
lrwxrwxrwx 1 young young    20 Sep 28 08:27 a.txt.soft.link -> d1/d1_1/d1_1_1/a.txt
```

User

useradd -d
useradd -m mat

useradd -u 1000 -g 2000 mat
useradd -g grp1, grp2 mat

userdel mat
userdel -r mat

passwd mat

groups mat
groupadd grp3
groupadd -g 1701 grp3
so groupadd

groupmod -g 491 grp3
groupmod -n grp4 grp3

groupdel grp3

SetUID Example

```
young/SysP$ vi fprn.c
young/SysP$ gcc fprn.c -o fprn
young/SysP$ chmod 4755 fprn
young/SysP$ ls -l fprn

                mat$ /home/young/SysP/fprn

young/SysP$ chmod 755 fprn
young/SysP$ ls -l fprn
                mat$ /home/young/SysP/fprn
```

```
#include <stdio.h>

void main()
{
   FILE *fp;

   // fp = fopen("/home/young/SysP/fprn.out", "w");
   fp = fopen("fprn.out", "w");

   if (fp != NULL) {
     printf("Hello, world!\n");
     fprintf(fp, "Hello, world!\n");
     fclose(fp);
   }
   else {
     error("Cannot open /home/young/SysP/fprn.out \n");
   }

}
```

Sticky Bit

```
young/SysP$ sudo useradd -m bob
young/SysP$ sudo useradd -m mat
young/SysP$ sudo addgroup sysp
young/SysP$ cat /etc/group
young/SysP$ id
young/SysP$ sudo usermod -g (gid of sysp) bob
young/SysP$ sudo usermod -g (gid of sysp) mat
young/SysP$ sudo cd /home/mat
young/SysP$ sudo mkdir SharedDir
young/SysP$ sudo chmod 1775 SharedDir

young/SysP$ sudo chown root:sysp SharedDir
```

```
bob$ cd /home/mat/SharedDir
bob$ vi bob.file
bob$ ls -al .

bob$ rm mat.file
```

```
mat$ cd SharedDir
mat$ vi mat.file
mat$ ls -al .

mat$ rm bob.file
```

Directory Handling (1)

```
#!/bin/bash

set -x # to display command and output

mkdir SysProg
mkdir SysProg/Lab
mkdir SysProg/Lab/L1
cd SysProg/
ls
mkdir HW
mkdir HW/H1
mkdir /home/young/Lab2/SysProg/HW/H2
pwd
cd HW
ls
rmdir H2
ls
cd ../../
pwd
ls
```

```
SysProg
├── HW
│   └── H1
└── Lab
    └── L1
```

Directory Handling (2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <errno.h>

int main() {
    char cwd[1024];

    // Create directories
    mkdir("SysProg", 0755);
    mkdir("SysProg/Lab", 0755);
    mkdir("SysProg/Lab/L1", 0755);

    // Change to SysProg directory
    chdir("SysProg");

    // List contents of SysProg
    system("ls");

    // Create HW directories
    mkdir("HW", 0755);
    mkdir("HW/H1", 0755);

    // Create absolute path directory
    mkdir("/home/young/Lab2/SysProg/HW/H2", 0755);

    // Print current working directory
    getcwd(cwd, sizeof(cwd));
    printf("Current directory: %s\n", cwd);
```

```c
// Change to HW directory
    chdir("HW");

    // List contents of HW
    system("ls");

    // Attempt to remove H2 (will fail
    // unless it's empty and accessible)
    if (rmdir("H2") != 0) {
        perror("rmdir H2");
    }

    // List contents again
    system("ls");

    // Go back to root of project
    chdir("../../");

    // Print current working directory
    getcwd(cwd, sizeof(cwd));
    printf("Current directory: %s\n", cwd);

    // List contents
    system("ls");

    return 0;
}
```

MS Copilot : converting d.bash into c code using c standard library

Directory Handling (3)

```
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

void list_dir(const char *path) {
    pid_t pid = fork();
    if (pid == 0) {
        execlp("ls", "ls", "-l", path, NULL);
        perror("execlp ls");
        exit(1);
    } else {
        wait(NULL);
    }
}

void print_pwd() {
    char buf[1024];
    if (getcwd(buf, sizeof(buf)) != NULL) {
        write(STDOUT_FILENO, buf, strlen(buf));
        write(STDOUT_FILENO, "\n", 1);
    } else {
        perror("getcwd");
    }
}
```

```
int main() {
    mkdir("SysProg", 0755);
    mkdir("SysProg/Lab", 0755);
    mkdir("SysProg/Lab/L1", 0755);

    chdir("SysProg");
    list_dir(".");

    mkdir("HW", 0755);
    mkdir("HW/H1", 0755);

    mkdir("/home/young/Lab2/SysProg/HW/H2", 0755);
    // May fail if path doesn't exist

    print_pwd();

    chdir("HW");
    list_dir(".");

    if (rmdir("H2") != 0) {
        perror("rmdir H2");
    }

    list_dir(".");

    chdir("../../");
    print_pwd();
    list_dir(".");

    return 0;
}
```

MS Copilot : converting d.bash into c code using c standard library

Directory Handling (2)

```
#!/bin/bash

set -x # to display command and output

mkdir SysProg
mkdir SysProg/Lab
mkdir SysProg/Lab/L1
cd SysProg/
ls
mkdir HW
mkdir HW/H1
mkdir /home/young/Lab2/SysProg/HW/H2
pwd
cd HW
ls
rmdir H2
ls
cd ../../
pwd
ls
```

```
SysProg
├── HW
│   └── H1
└── Lab
    └── L1
```

Without **x** permission

You cannot access files inside the directory
- even if you know their names.

You'll get "Permission denied" errors for most operations.

MS Copilot : setuid

Stat needs **x** permission

To find the state of a file inside a directory
- meaning to check its existence, size, type, or metadata
- you need execute (x) permission on the directory.

Permission	Why It's Needed
x (execute)	Allows you to access the file by name inside the directory
r (read)	Allows you to list the directory contents (e.g., with ls)

To check the state of a file inside a directory:

You must have x permission on the directory

r permission is only needed if you want to list files

MS Copilot : setuid

Directory Permission

r	list	Allows listing the contents (e.g., **ls**)
w	modify	Allows creating, deleting, or renaming files inside
x	enter	Allows entering the directory and accessing (read, write, stat) files inside by name

rwx	7	can list, modify, enter/access
rw-	6	can list, modify, but not modify or enter/access
r-x	5	can list, enter/access, but not modify
-wx	3	can modify, enter/access, but not list
r--	4	can list contents, but not enter/access or modify
-w-	2	can not list or modify or enter/access
--x	1	can enter/access if they know file names, but cannot list or modify

MS Copilot : setuid

Directory Permission

No x — cannot enter the directory or access files inside it even if you know their names.

r only — can list the directory contents (e.g., with **ls**), but cannot open or stat any files inside.

x only — can access files by name if you know them, but cannot list the directory contents.

No x	r only	x only	
rwx	rwx	rwx	can list, modify, enter
rw-	rw-	rw-	can list, but not modify or enter
r-x	r-x	r-x	can list, enter, but not modify
-wx	-wx	-wx	can modify, enter, but not list
r--	r--	r--	can list contents, but not enter or modify
-w-	-w-	-w-	can not list or modify or enter
--x	--x	--x	can enter if they know file names, but cannot list or modify

MS Copilot : setuid

Directory Permission

r	listing (ls) the directory	rx	listing (ls -l) files inside
w		wx	creating, deleting, or renaming files inside
x	entering the directory	x	accessing (read. write, stat) files inside by name

rwx	7	can list, modify, enter
rw-	6	can list, but not modify or enter
r-x	5	can list, enter, but not modify
-wx	3	can modify, enter, but not list
r--	4	can list contents, but not enter or modify
-w-	2	can not list or modify or enter
--x	1	can enter if they know file names, but cannot list or modify

MS Copilot : setuid

Commands That Use **r** Permission on a Directory

When a directory has read (**r**) permission,
it allows the user to list the names
of files and subdirectories inside it
- but not necessarily access their contents.

Command	What It Does	Requires w on Directory
ls mydir	Lists the contents of the directory	
find mydir	Searches through the dir tree	
tar -cf archive.tar mydir	Archives the directory	
du mydir	Shows disk usage of the directory	
stat mydir/*	Displays metadata of files inside	
grep "text" mydir/*	Searches files by name pattern	

MS Copilot : setuid

To access file contents

Read (**r**) permission alone is not enough to access file contents

- you also need execute (**x**) permission
  to enter the directory and open files.

Without **x**, you can list file names but cannot access or open them

MS Copilot : setuid

Commands That Use **w** Permission on a Directory

When a directory has write (**w**) permission,
it allows the user to modify the contents of that directory

- meaning they can create, delete, or rename files and subdirectories inside it.

Command	What It Does	Requires w on Directory
touch newfile.txt	Creates a new file	
rm file.txt	Deletes a file	
mv old.txt new.txt	Renames a file	
cp file.txt dir/	Copies a file into the directory	
mkdir newdir	Creates a subdirectory	
rmdir subdir	Removes an empty subdirectory	
echo "text" > file.txt	Writes to a file (if creating it)	
truncate -s 0 file.txt	Empties a file (if creating it)	

MS Copilot : setuid

To access or modify file contents

rite (w) permission alone is not enough to access or modify files

- you also need execute (x) permission
  to enter the directory and access its contents.

Without x, you can't even rm or touch inside the directory,
even if w is set.

MS Copilot : setuid

Commands That Use **x** Permission on a Directory

When a directory has execute (**x**) permission,
it allows a user to enter the directory and access files by name
- even if they can't list its contents.

Command	What It Does	Requires w on Directory
cd mydir	Enter the directory	
cat mydir/file.txt	Read a file inside (if you know the name)	
rm mydir/file.txt	Delete a file inside	
mv mydir/file.txt .	Move a file inside out	
touch mydir/newfile.txt	Create a file inside	
chmod +x mydir/file.sh	Change permissions of a file inside	

MS Copilot : setuid

Directory Access Permissions (1)

```
#!/bin/bash

function init() {
  if [ -d "dir" ]; then
     chmod 700 dir
     rm -fr dir
  fi
  mkdir dir
  chmod 700 dir
  cp file dir
  ls -ld dir
  ls -l dir/file
}
```

```
function check() {

  echo "-----------------------------------"
  echo "checking r : list"
  echo "ls dir"
  ls dir

  echo "-----------------------------------"
  echo "checking rx : list in long format"
  echo "ls -l dir"
  ls -l dir

  echo "-----------------------------------"
  echo "checking wx : modify"
  echo "rm  dir/file"
  rm dir/file && cp file dir

  echo "-----------------------------------"
  echo "checking x : enter"
  echo "cd dir "
  cd dir && cd ..

  echo "-----------------------------------"
  echo "checking x : read access"
  echo "cat dir/file "
  cat dir/file

  echo "-----------------------------------"
  echo "checking x : write access"
  echo "echo 'This is a file.' >  dir/file "
  echo "This is a file." >  dir/file

  echo "-----------------------------------"
  echo "checking x : stat"
  echo "stat dir/file"
  stat dir/file

  echo " "
}
```

Directory Access Permissions (2)

```
echo "==================================="
echo "700 drwx------ dir"
echo "==================================="
init
chmod 700 dir
check

echo "==================================="
echo "600 drw------- dir"
echo "==================================="
init
chmod 600 dir
check

echo "==================================="
echo "500 dr-x------ dir"
echo "==================================="
init
chmod 500 dir
check

echo "==================================="
echo "300 d-wx------ dir"
echo "==================================="
init
chmod 300 dir
check

echo "==================================="
echo "400 dr-------- dir"
echo "==================================="
init
chmod 400 dir
check
```

```
echo "==================================="
echo "200 d-w------- dir"
echo "==================================="
init
chmod 200 dir
check

echo "==================================="
echo "100 d--x------ dir"
echo "==================================="
init
chmod 100 dir
check
```

Directory Access Permissions (3)

```
==================================
700 drwx------ dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
file
----------------------------------
checking rx : list in long format
ls -l dir
total 4
-rw-rw-r-- 1 young young 16 Oct 19 00:47 file
----------------------------------
checking wx : modify
rm  dir/file
----------------------------------
checking x : enter
cd dir
----------------------------------
checking x : read access
cat dir/file
This is a file.
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
----------------------------------
checking x : stat
stat dir/file
  File: dir/file
  Size: 16              Blocks: 8          IO Block: 4096   regular file
Device: 10302h/66306d   Inode: 1054697     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   young)   Gid: ( 1000/   young)
Access: 2025-10-19 00:47:01.930756891 +0900
Modify: 2025-10-19 00:47:01.930756891 +0900
Change: 2025-10-19 00:47:01.930756891 +0900
 Birth: 2025-10-19 00:47:01.926756891 +0900
```

```
==================================
600 drw------- dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
file
----------------------------------
checking rx : list in long format
ls -l dir
ls: cannot access 'dir/file': Permission denied
total 0
-????????? ? ? ? ?            ? file
----------------------------------
checking wx : modify
rm  dir/file
rm: cannot remove 'dir/file': Permission denied
----------------------------------
checking x : enter
cd dir
run: line 35: cd: dir: Permission denied
----------------------------------
checking x : read access
cat dir/file
cat: dir/file: Permission denied
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
run: line 45: dir/file: Permission denied
----------------------------------
checking x : stat
stat dir/file
stat: cannot statx 'dir/file': Permission denied
```

Directory Access Permissions (4)

```
==================================
500 dr-x------ dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
file
----------------------------------
checking rx : list in long format
ls -l dir
total 4
-rw-rw-r-- 1 young young 16 Oct 19 00:47 file
----------------------------------
checking wx : modify
rm  dir/file
rm: cannot remove 'dir/file': Permission denied
----------------------------------
checking x : enter
cd dir
----------------------------------
checking x : read access
cat dir/file
This is a file.
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
----------------------------------
checking x : stat
stat dir/file
  File: dir/file
  Size: 16              Blocks: 8          IO Block: 4096   regular file
Device: 10302h/66306d   Inode: 1054697     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   young)   Gid: ( 1000/   young)
Access: 2025-10-19 00:47:01.946756892 +0900
Modify: 2025-10-19 00:47:01.946756892 +0900
Change: 2025-10-19 00:47:01.946756892 +0900
 Birth: 2025-10-19 00:47:01.942756892 +0900
```

```
==================================
300 d-wx------ dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
ls: cannot open directory 'dir': Permission denied
----------------------------------
checking rx : list in long format
ls -l dir
ls: cannot open directory 'dir': Permission denied
----------------------------------
checking wx : modify
rm  dir/file
----------------------------------
checking x : enter
cd dir
----------------------------------
checking x : read access
cat dir/file
This is a file.
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
----------------------------------
checking x : stat
stat dir/file
  File: dir/file
  Size: 16              Blocks: 8          IO Block: 4096
Device: 10302h/66306d   Inode: 1054697     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   young)   Gid: ( 10
Access: 2025-10-19 00:47:01.954756892 +0900
Modify: 2025-10-19 00:47:01.954756892 +0900
Change: 2025-10-19 00:47:01.954756892 +0900
 Birth: 2025-10-19 00:47:01.954756892 +0900
```

Directory Access Permissions (5)

```
==================================
400 dr-------- dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
file
----------------------------------
checking rx : list in long format
ls -l dir
ls: cannot access 'dir/file': Permission denied
total 0
-????????? ? ? ? ?            ? file
----------------------------------
checking wx : modify
rm  dir/file
rm: cannot remove 'dir/file': Permission denied
----------------------------------
checking x : enter
cd dir
run: line 35: cd: dir: Permission denied
----------------------------------
checking x : read access
cat dir/file
cat: dir/file: Permission denied
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
run: line 45: dir/file: Permission denied
----------------------------------
checking x : stat
stat dir/file
stat: cannot statx 'dir/file': Permission denied
```

```
==================================
200 d-w------- dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
ls: cannot open directory 'dir': Permission denied
----------------------------------
checking rx : list in long format
ls -l dir
ls: cannot open directory 'dir': Permission denied
----------------------------------
checking wx : modify
rm  dir/file
rm: cannot remove 'dir/file': Permission denied
----------------------------------
checking x : enter
cd dir
run: line 35: cd: dir: Permission denied
----------------------------------
checking x : read access
cat dir/file
cat: dir/file: Permission denied
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
run: line 45: dir/file: Permission denied
----------------------------------
checking x : stat
stat dir/file
stat: cannot statx 'dir/file': Permission denied
```

Directory Access Permissions (6)

```
==================================
100 d--x------ dir
==================================
drwx------ 2 young young 4096 Oct 19 00:47 dir
-rw-rw-r-- 1 young young 16 Oct 19 00:47 dir/file
----------------------------------
checking r : list
ls dir
ls: cannot open directory 'dir': Permission denied
----------------------------------
checking rx : list in long format
ls -l dir
ls: cannot open directory 'dir': Permission denied
----------------------------------
checking wx : modify
rm  dir/file
rm: cannot remove 'dir/file': Permission denied
----------------------------------
checking x : enter
cd dir
----------------------------------
checking x : read access
cat dir/file
This is a file.
----------------------------------
checking x : write access
echo 'This is a file.' >  dir/file
----------------------------------
checking x : stat
stat dir/file
  File: dir/file
  Size: 16              Blocks: 8          IO Block: 4096   regular file
Device: 10302h/66306d   Inode: 1054697     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   young)   Gid: ( 1000/   young)
Access: 2025-10-19 00:47:01.970756893 +0900
Modify: 2025-10-19 00:47:01.970756893 +0900
Change: 2025-10-19 00:47:01.970756893 +0900
 Birth: 2025-10-19 00:47:01.966756893 +0900
```

setuid (1)

setuid is a special permission bit
that allows a user to run an executable
with the permissions of the file owner,
rather than the permissions
of the user who launched it.

What Setuid Does
Normally, when you run a program,
it executes with your user privileges.

If a file has the setuid bit set,
it runs with the owner's privileges
-- often root.

to check setgid bit is set,
use **ls -l** or ls -ld:

The s or S in the group's execute position
indicates setgid is active

S indicates no exec permission x
s indicates exec permission x and setuid bit set

```
-rw-rw-r--
-rwxr-xr-x
```

```
-rwS-rw-r--
-rwsr-xr-x
```

MS Copilot : setuid

setuid (2)

Example

1. Check if setuid is set:

ls -l /usr/bin/passwd

-rwsr-xr-x 1 root root 54256 Oct 18 10:00 /usr/bin/passwd

The s in rws means setuid is active.

This allows regular users to change their passwords,
even though /etc/shadow is only writable by root.

2. Set the setuid bit:

sudo chmod u+s myprogram

Adds setuid to myprogram

3. Remove the setuid bit:

sudo chmod u-s myprogram

Security Warning

Setuid can be dangerous if misused:

it can allow privilege escalation
Should only be used on trusted, secure binaries

MS Copilot : setuid

Testing setuid (1)

```
// setuid_test1.c


#include <stdio.h>
#include <unistd.h>

int main(void){

  printf("---------------------------\n");
  printf("test1 \n");
  printf("login : %s \n", getlogin());
  printf("uid   : %d \n", getuid());
  printf("euid  : %d \n", geteuid());

}
```

```
// setuid_test2.c


#include <stdio.h>
#include <unistd.h>

int main(void){

  FILE *fp;
  char buf[256];

  printf("---------------------------\n");
  printf("test2 \n");
  printf("login : %s \n", getlogin());
  printf("uid   : %d \n", getuid());
  printf("euid  : %d \n", geteuid());

  fp = fopen("/home/young/my.txt", "r");

  if (fp == NULL) {
    perror("Error in opening");
    return 1;
  }

  if (fgets(buf, sizeof(buf), fp) != NULL) {
     printf("Read string: %s\n", buf);
  } else {
     printf("No string read or file is empty.\n");
  }

  fclose(fp);

}
```

Testing setuid (2)

```
#!/bin/bash
echo "=============================="
echo "user young"
echo "=============================="

echo "* making the executable test1 from setuid_test1.c"
echo "* making the executable test2 from setuid_test2.c"

gcc -Wall -o test1 setuid_test1.c
gcc -Wall -o test2 setuid_test2.c

echo "* creating my.txt"

echo -n "This is my text." > my.txt

echo "* making access permission drwxr-xr-x for ~/"
echo "* making access permission -rwxr-xr-x for ~/test1"
echo "* making access permission -rwxr-xr-x for ~/test2"
echo "* making access permission -rwx------ for ~/my.txt"

chmod go+rx  /home/young
chmod go+rx  /home/young/test1
chmod go+rx  /home/young/test2
chmod go-rwx /home/young/my.txt

echo "* checking access permissions"

ls -ld /home/young
ls -l  /home/young/test1
ls -l  /home/young/test2
ls -l  /home/young/my.txt

echo "========================================="
echo "* young is executing /home/young/test1"
echo "* young is executing /home/young/test2"
echo "========================================="

/home/young/test1
/home/young/test2
```

```
echo "========================================="
echo "user1 executes young's -rwxr-xr-x test1"
echo "user1 executes young's -rwxr-xr-x test2"
echo "========================================="

echo "* user1 executing /home/young/test1"
echo "* user1 executing /home/young/test2"

su - user1 -c '/home/young/test1'
su - user1 -c '/home/young/test2'


echo "========================================="
echo "young sets setuid to test1"
echo "young sets setuid to test2"
echo "========================================="

chmod 4755 /home/young/test1
chmod 4755 /home/young/test2

ls -l /home/young/test1
ls -l /home/young/test1

echo "========================================="
echo "user1 executes young's -rwsr-xr-x test1"
echo "user1 executes young's -rwsr-xr-x test2"
echo "========================================="

echo "* user1 executing /home/young/test1"
echo "* user1 executing /home/young/test2"

su - user1 -c /home/young/test1
su - user1 -c /home/young/test2
```

Testing setuid (3)

```
==============================
user young
==============================
* making the executable test1 from setuid_test1.c
* making the executable test2 from setuid_test2.c
* creating my.txt
* making access permission drwxr-xr-x for ~/
* making access permission -rwxr-xr-x for ~/test1
* making access permission -rwxr-xr-x for ~/test2
* making access permission -rwx------ for ~/my.txt
* checking access permissions
drwxr-xr-x 21 young young 4096 Oct 15 17:40 /home/young
-rwxrwxr-x 1 young young 16144 Oct 15 17:40 /home/young/test1
-rwxrwxr-x 1 young young 16360 Oct 15 17:40 /home/young/test2
-rw------- 1 young young 16 Oct 15 17:40 /home/young/my.txt
========================================
* young is executing /home/young/test1
* young is executing /home/young/test2
========================================
--------------------------
test1
login : young
uid   : 1000
euid  : 1000
--------------------------
test2
login : young
uid   : 1000
euid  : 1000
Read string: This is my text.

========================================
user1 executes young's -rwxr-xr-x test1
user1 executes young's -rwxr-xr-x test2
========================================
* user1 executing /home/young/test1
* user1 executing /home/young/test2
--------------------------
test1
login : young
uid   : 1001
euid  : 1001
--------------------------
test2
login : young
uid   : 1001
euid  : 1001
Error in opening: Permission denied
========================================
young sets setuid to test1
young sets setuid to test2
========================================
-rwsr-xr-x 1 young young 16144 Oct 15 17:40 /home/young/test1
-rwsr-xr-x 1 young young 16144 Oct 15 17:40 /home/young/test1
====================================
user1 executes young's -rwsr-xr-x test1
user1 executes young's -rwsr-xr-x test2
====================================
* user1 executing /home/young/test1
* user1 executing /home/young/test2
--------------------------
test1
login : young
uid   : 1001
euid  : 1000
--------------------------
test2
login : young
uid   : 1001
euid  : 1000
Read string: This is my text.
```

Group (1)

managing groups is done using commands like
groupadd, **groupdel**, **usermod**, and **gpasswd**.

Add a Group

sudo groupadd developers

Creates a new group called developers

Remove a Group

sudo groupdel developers

Deletes the group developers

Only works if no users are assigned to it
as their primary group

MS Copilot : group

Group (2)

Add a User to a Group

sudo usermod -aG developers alice

Adds user alice to the developers group

-aG means "append to supplementary groups"

Remove a User from a Group

Edit the /etc/group file manually
wsing **nano** editor

sudo nano /etc/group

Find the line:
developers:x:1001:alice,bob

Remove the username (alice) from the list:

developers:x:1001:bob
Save and exit.

MS Copilot : group

usermod (1)

used to modify an existing user account

It's a powerful tool for changing user details like group membership, home directory, login shell, and more.

sudo usermod [options] username

Add User to a Group

sudo usermod -aG developers alice

adds alice to the developers group

-a appends the group;
-G specifies the group list

Change Home Directory

sudo usermod -d /home/newhome alice

changes alice's home directory to /home/newhome

sudo usermod -d /home/newhome -m alice

moves files as well

Change Login Shell

sudo usermod -s /bin/zsh alice

sets zsh as the default shell for alice

MS Copilot : usermod

usermod (2)

Lock or Unlock Account

sudo usermod -L alice # Lock
sudo usermod -U alice # Unlock

temporarily disabling the ability to log in.

Change Primary Group

sudo usermod -g staff alice

sets staff as alice's primary group

View Changes

after modifying, you can verify with:

id alice

Or check /etc/passwd and /etc/group.

MS Copilot : usermod

touch (1)

The touch command in Linux is used
to create empty files or
update timestamps of existing ones.

Basic Usage

touch filename.txt

If filename.txt doesn't exist,
it creates an empty file.

If it does exist,
it updates the file's modification time
to the current time.

MS Copilot : touch

touch (2)

Create Multiple Files

touch file1.txt file2.txt file3.txt

Update Timestamp Only

touch existing.txt

Changes the modification time
without altering the content.

Set a Specific Timestamp

touch -t 202510201200 filename.txt

Sets the timestamp to Oct 20, 2025 at 12:00 PM

Permissions Required

To use touch:

on the directory,
you need write (w) and execute (x) permissions

on the file (if it exists)
you need write (w) permission

MS Copilot : touch

Setgid

setgid (Set Group ID) is a special permission bit
- affects both files and directories,
- behaves differently depending on the context.

to check setgid bit is set,
use **ls -l** or ls -ld:

The s or S in the group's execute position
indicates setgid is active

S indicates no exec permission x
s indicates exec permission x and setgid bit set

```
drwxrw-r--        drwxrwSr--
drwxrwxr-x        drwxrwsr-x
```

```
-rw-rw-r--        -rw-rwSr--
-rw-r-xr-x        -rw-r-sr-x
```

MS Copilot : setuid

Setgid on directories (1)

setgid on directories

New files and subdirectories created inside
inherit the group of the directory,
rather than the user's default group.

This is useful for collaborative directories
where you want all content to stay
within a shared group.

chmod g+s shared_folder
or
chmod 2755 shared_folder

g+s sets the setgid bit.
2 in **2755** sets the setgid bit.

any file created in **shared_folder**
will have the same group as **shared_folder**.

MS Copilot : setuid

Setgid on directories (2)

If a directory has the setgid bit
and is group writable,
if a user (not the directory owner)
with a different primary group
writes a file in that directory
and has supplemental membership
in the group that owns the directory,
the new file gets the same group ownership
as the directory.

Not the primary group of the user writing the file.

https://unix.stackexchange.com/questions/369611/meaning-of-setgid-on-an-executable

Setgid on directories (3)

As an example we have two users,
foo and **bar**.
foo's primary group is also **foo**.
bar's primary group is **bar**,

but **bar** is a supplemental member of **foo**.

foo@valhalla:$ id
uid=1002(foo) gid=1002(foo) groups=1002(foo)

bar@valhalla:~$ id
uid=1003(bar) gid=1003(bar) groups=1003(bar),1002(foo)

foo@valhalla:~$ grep foo /etc/group
foo:x:1002:bar

foo@valhalla:~$ grep bar /etc/group
foo:x:1002:bar
bar:x:1003:

https://unix.stackexchange.com/questions/369611/meaning-of-setgid-on-an-executable

Setgid on directories (4)

foo creates a directory /tmp/foodir and
make it setgid and group writable.

foo@valhalla:~$ mkdir /tmp/foodir
foo@valhalla:~$ chmod g+ws /tmp/foodir
foo@valhalla:~$ ls -ld /tmp/foodir
drwxrwsr-x 2 foo foo 4096 Jun 6 19:30 /tmp/foodir

now **bar** touches a file barfile in /tmp/foodir as the user bar.

bar@valhalla:~$ touch /tmp/foodir/barfile
bar@valhalla:~$ ls -l /tmp/foodir/barfile
-rw-r--r-- 1 bar foo 0 Jun 6 19:32 /tmp/foodir/barfile

Notice the group ownership of /tmp/foodir/barfile is foo,
not bar which is the user **bar**'s primary group.

https://unix.stackexchange.com/questions/369611/meaning-of-setgid-on-an-executable

Setgid on directories (6)

Note that **foo** isn't a member of group **bar**.
but **bar** is a supplemental member of **foo**.

bar@valhalla:~$ mkdir /tmp/bardir
bar@valhalla:~$ chmod g+ws /tmp/bardir
bar@valhalla:~$ ls -ld /tmp/bardir
drwxrwsr-x 2 bar bar 4096 Jun 6 19:34 /tmp/bardir

when **foo** touch a file foofile, a permission error.

foo@valhalla:~$ touch /tmp/bardir/foofile
touch: cannot touch '/tmp/bardir/foofile': Permission denied

https://unix.stackexchange.com/questions/369611/meaning-of-setgid-on-an-executable

Setgid on directories (7)

when **foo** touch a file foofile, a permission error.

foo@valhalla:~$ touch /tmp/bardir/foofile
touch: cannot touch '/tmp/bardir/foofile': Permission denied

bar touch’es a file in /tmp (a not setgid directory that bar can write to)

bar@valhalla:~$ touch /tmp/barfile
bar@valhalla:~$ ls -ld /tmp/barfile
-rw-r--r-- 1 bar bar 0 Jun 6 19:36 /tmp/barfile
The owner and group are both bar.

https://unix.stackexchange.com/questions/369611/meaning-of-setgid-on-an-executable

Setgid on files (1)

setgid on files

The process runs with the group ID of the file,
not the group of the user who launched it.

similar to setuid,
but applies to group privileges.

chmod g+s myprogram
or
chmod 2755 myprogram

g+s sets the setgid bit.
2 in **2**755 sets the setgid bit.

If **myprogram** is owned by group **admin**,
it runs with **admin** group privileges.

MS Copilot : setuid

Setgid on files (2)

ls -l my_bin
r-xr-s--- root wheel my_bin

setuid bit is off but setgid bit is on
for the executable my_bin

The setgid bit works the same as the setuid bit,
but for the group ID.

So the process will be run
with an effective group ID of wheel.

The effective (and real) user ID will still be
that of whichever user started the program.

your user's membership in that group
doesn't matter one way or the other.

user userA need not be a a member of group wheel.

even if userA DIDNOT belong to wheel group
they would still get the effective group ID of wheel
because the setgid is set

The program would be running
with a real & effective user id of userA,
a real group ID of (user A's group),
and an effective group ID of wheel.

Whether the user can obtain group wheel
(and not just that one program)
depends on how secure that one program is.

Normally the goal is for the program
to not let the user obtain the elevated privilege

MS Copilot : setuid

Setgid on files (2)

ls -l my_bin
r-xr-s--- root wheel my_bin

setuid bit is off but setgid bit is on
for the executable my_bin

The setgid bit works the same as the setuid bit,
but for the group ID.

So the process will be run
with an effective group ID of wheel.

The effective (and real) user ID will still be
that of whichever user started the program.

your user's membership in that group
doesn't matter one way or the other.

user userA need not be a a member of group wheel.

even if userA DIDNOT belong to wheel group
they would still get the effective group ID of wheel
because the setgid is set

The program would be running
with a real & effective user id of userA,
a real group ID of (user A's group),
and an effective group ID of wheel.

Whether the user can obtain group wheel
(and not just that one program)
depends on how secure that one program is.

Normally the goal is for the program
to not let the user obtain the elevated privilege

MS Copilot : setuid

Setgid (1)

setgid (Set Group ID) is a special permission bit
- affects both files and directories,
- behaves differently depending on the context.

```
* user1 executing /home/young/test1
* user1 executing /home/young/test2
--------------------------
test1
login : young
uid   : 1001
euid  : 1001
--------------------------
test2
login : young
uid   : 1001
euid  : 1001
Error in opening: Permission denied
========================================
young sets setuid to test1
young sets setuid to test2
========================================
-rwsr-xr-x 1 young young 16144 Oct 15 17:40 /home/young/test1
-rwsr-xr-x 1 young young 16144 Oct 15 17:40 /home/young/test1
======================================
user1 executes young's -rwsr-xr-x test1
user1 executes young's -rwsr-xr-x test2
======================================
* user1 executing /home/young/test1
* user1 executing /home/young/test2
--------------------------
test1
login : young
uid   : 1001
euid  : 1000
--------------------------
test2
login : young
uid   : 1001
euid  : 1000
Read string: This is my text.
```

MS Copilot : converting d.bash into c code using c standard library

Sticky bit (1)

the sticky bit is
a special permission set on directories
that restricts file deletion:

only the file owner,
the directory owner,
or root can delete or rename files
inside that directory

- even if others have write access.

It’s commonly used on shared directories
to prevent users from deleting each other’s files.

.

Example: /tmp Directory

ls -ld /tmp

drwxrwxrwt 10 root root 4096 Oct 23 10:00 /tmp

The t at the end of the permissions (rwxrwxrwt)
indicates the sticky bit is set.

MS Copilot : sticky bit

Sticky bit (2)

How to Set the Sticky Bit

sudo chmod +t shared_dir

Adds sticky bit to shared_dir

To remove it:

sudo chmod -t shared_dir

Why It Matters

Without the sticky bit:

Any user with write access
to the directory **/tmp**
can delete any file inside it

With the sticky bit:

Users can only delete their own files

MS Copilot : sticky bit

Sticky bit example (1)

Create a directory and provide all the users
read-write-execute access to it :

 # mkdir allAccess

 # chmod 777 allAccess/

 # ls -ld allAccess/
drwxrwxrwx 2 himanshu himanshu 4096 Oct 24 15:43 allAccess/

Now, create multiple files in this directory (with different users)
such that all users have read-write-execute access to them.

 # ls -l allAccess/
total 0
-rwxrwxrwx 1 himanshu himanshu 0 Oct 24 15:48 user1
-rwxrwxrwx 1 guest guest 0 Oct 24 16:11 user_file_0
-rwxrwxrwx 1 guest-2 guest-2 0 Oct 24 16:15 user_file_1

https://www.thegeekstuff.com/2013/02/sticky-bit/

Sticky bit example (2)

```
-rwxrwxrwx 1 himanshu  himanshu  0 Oct 24 15:48 user1
-rwxrwxrwx 1 guest     guest     0 Oct 24 16:11 user_file_0
-rwxrwxrwx 1 guest-2   guest-2   0 Oct 24 16:15 user_file_1
```

The files user_file_0 and user_file_1
are created by different users
but have read-write-execute access on for all the users.

This means that the user ‘guest’
can delete or rename the file created by user ‘guest-2’.

In order to avoid this,
sticky bit can be set on the directory allAccess.

https://www.thegeekstuff.com/2013/02/sticky-bit/

Sticky bit example (3)

Now, turn ON the sticky bit on the directory
by using +t flag of chmod command.

 # chmod +t allAccess/

 # ls -ld allAccess/
drwxrwxrwt 2 himanshu himanshu 4096 Oct 24 16:19 allAccess/

As can be observed, a permission bit ‘t’
is introduced in the permission bits of the directory.

https://www.thegeekstuff.com/2013/02/sticky-bit/

Sticky bit example (4)

```
-rwxrwxrwx 1 himanshu  himanshu  0 Oct 24 15:48 user1
-rwxrwxrwx 1 guest     guest     0 Oct 24 16:11 user_file_0
-rwxrwxrwx 1 guest-2   guest-2   0 Oct 24 16:15 user_file_1
```

Now, if the user ‘guest’ tries to rename the file ‘user_file_1’,
here is what happens :

```
$ mv allAccess/user_file_1 allAccess/user_file_0
mv: cannot move `/home/himanshu/allAccess/user_file_1'
to `/home/himanshu/allAccess/user_file_0': Operation not permitted
```

So we see that the operation was not permitted.

https://www.thegeekstuff.com/2013/02/sticky-bit/

Sticky bit example II (1)

Here's a practical example showing how file deletion behaves in a directory with both the sticky bit and setgid set.

Setup: Create a Shared Directory

sudo mkdir /shared
sudo chown root:developers /shared
sudo chmod 3775 /shared

3 → sets both setgid and sticky bit
775 → read/write/execute for owner and group, read/execute for others

Group developers must exist, and users should be added to it

Add Users to the Group

sudo usermod -aG developers alice
sudo usermod -aG developers bob

MS Copilot : removing file example in a directory with sticky bit and setgid set

Sticky bit example II (2)

Behavior in /shared

1. Alice creates a file:

sudo -u alice touch /shared/alice.txt

2. Bob tries to delete Alice's file:

sudo -u bob rm /shared/alice.txt

Permission denied —
because of the sticky bit,
Bob cannot delete Alice's file
even though they share group access.

What Each Bit Does

Bit	Effect
Sticky bit (+t)	Only file owner, directory owner, or root can delete files
Setgid (g+s)	New files inherit the directory's group (developers)

Who Can Delete

User	Can Delete alice.txt?
Alice (file owner)	Yes
Root	Yes
Bob (same group)	No
Directory owner	Yes

MS Copilot : removing file example in a directory with sticky bit and setgid set

chown (1)

The chown command is used to change
the ownership of files and directories
- specifically the user and/or group
  that owns them.

Basic Syntax

sudo chown [OPTIONS] user[:group] file/dir

user → new owner
group → new group (optional)

If you omit group, only the user changes

Common Examples

Change Owner Only

sudo chown alice report.txt

Makes alice the owner of report.txt

Change Owner and Group

sudo chown alice:developers report.txt

Sets alice as owner and developers as group

Change Ownership Recursively

sudo chown -R alice:developers /project

Applies ownership change
to /project and all its contents

MS Copilot : chown

chown (1)

The chown command is used to change
the ownership of files and directories
- specifically the user and/or group
  that owns them.

Basic Syntax

sudo chown [OPTIONS] user[:group] file/dir

user → new owner
group → new group (optional)

If you omit group, only the user changes

Common Examples

Change Owner Only

sudo chown alice report.txt

Makes alice the owner of report.txt

Change Owner and Group

sudo chown alice:developers report.txt

Sets alice as owner and developers as group

Change Ownership Recursively

sudo chown -R alice:developers /project

Applies ownership change
to /project and all its contents

MS Copilot : chown

chown (2)

View Ownership

ls -l

Shows owner and group
in the third and fourth columns

Permissions Required

Only root or the current owner
can change ownership

You cannot give ownership
to another user unless you're root

MS Copilot : chown

sudo (1)

The sudo command stands for "superuser do"
- it allows a permitted user to run commands
with elevated (root) privileges,
temporarily acting as the system administrator.

Basic Syntax

sudo command [arguments]

Runs the specified command as root
(or another user, if specified)

Common Examples

Install Software
sudo apt install nginx

Edit System Files
sudo nano /etc/hosts

Add a New User
sudo useradd newuser

Reboot the System
sudo reboot

MS Copilot : sudo

sudo (2)

How It Works

When you run sudo,
you're prompted for your own password, not root's

Your access is controlled by the /etc/sudoers file

After successful use,
you get a grace period (usually 5 minutes)
where you don’t need to re-enter your password

Use with Caution

sudo gives you full control
- including the ability to break the system

Always double-check commands
before running them with sudo

MS Copilot : sudo

sudo (3)

Only users who are explicitly granted permission can use sudo in Linux.

This is controlled by the /etc/sudoers file and related configuration.

Who Can Use sudo

User Type	Can Use sudo?	How
Root user	Yes	Always has full access
Users in sudo or wheel group	Yes	Automatically allowed via group membership
Other users	No	Must be manually added to sudoers or a privileged group

Use with Caution

sudo gives you full control
- including the ability to break the system

Always double-check commands before running them with sudo

MS Copilot : sudo

sudo (4)

How to Grant sudo Access

1. Add user to sudo group (Debian/Ubuntu):
sudo usermod -aG sudo username

2. Add user to wheel group (RHEL/CentOS/Fedora):
sudo usermod -aG wheel username

3. Edit /etc/sudoers directly (with caution):
sudo visudo

Add a line like:
username ALL=(ALL) ALL

Check if a User Has sudo Access

sudo -l -U username

Lists allowed commands for that user

MS Copilot : sudo

References

[1] http://minix1.woodhull.com/current/2.0.4/
[2]