CS2211b

Software Tools and Systems Programming

Week 2b

SCP & File Permissions

Announcements

Assignment 1 Posted!

Transferring Files: `scp`

Secure Copy (`scp`)

With Command Line (UNIX-Like)

We can copy files to and from a remote server using the `scp` command:

```
scp [[user@]host1:]file1 ... [[user@]host2:]file2
```

Secure Copy (scp)

With Command Line (UNIX-Like)

We can copy files to and from a remote server using the `scp` command:

```
scp [[user@]host1:]file1 ... [[user@]host2:]file2
```

The origin

The destination

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 1: Copy a file from the local machine to the remote course server.

```
scp myfile.c dservos5@cs2211b.gaul.csd.uwo.ca:myfile.c
```

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 1: Copy a file from the local machine to the remote course server.

```
scp myfile.c dservos5@cs2211b.gaul.csd.uwo.ca:myfile.c
```

Local File
(in CWD)

User Name

Remote Hostname

Remote File
(Will be created in CWD on server)

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 2: Copy the local file, *file1*, from ~/dir1/file1 to ~/dirA/fileA on the course server.

```
scp ~/dir1/file1 dservos5@cs2211b.gaul.csd.uwo.ca:~/dirA/fileA
```

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 2: Copy the local file, *file1*, from ~/dir1/file1 to ~/dirA/fileA on the course server.

```
scp ~/dir1/file1 dservos5@cs2211b.gaul.csd.uwo.ca:~/dirA/fileA
```

Local Path

- Could use relative path if CWD is ~:
  dir1/file1

Remote Path

- dirA has to exist
- Could use relative path:
  dirA/fileA
- In most cases remote CWD is ~ by default

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 3: Copy the local files, *a*, *b* and *c* in your home directory to *~/dirA* on the course server.

```
scp a b c dservos5@cs2211b.gaul.csd.uwo.ca:~/dirA
```

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 3: Copy the local files, *a*, *b* and *c* in your home directory to *~/dirA* on the course server.

```
scp a b c dservos5@cs2211b.gaul.csd.uwo.ca:~/dirA
```

We can list multiple files to copy…

but only if the destination is a directory

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 4: Copy the local directory *mydir* and all of its contents to the course server.

```
scp –r mydir dservos5@cs2211b.gaul.csd.uwo.ca:~
```

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 4: Copy the local directory *mydir* and all of its contents to the course server.

```
scp -r mydir dservos5@cs2211b.gaul.csd.uwo.ca:~
```

Need to use -r (recursive) option, just like with cp.

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 5: Copy *~/myscript.sh* from the course server to the local machine.

```
scp dservos5@cs2211b.gaul.csd.uwo.ca:myscript.sh .
```

Secure Copy (scp)

With Command Line (UNIX-Like)

Example 5: Copy *~/myscript.sh* from the course server to the local machine.

```
scp dservos5@cs2211b.gaul.csd.uwo.ca:myscript.sh .
```

Remote file now comes first as it is the file being copied (just like in cp).

Destination is just . (single dot). Means CWD.

Secure Copy (scp)

With Command Line (UNIX-Like)

- Can use wildcards with scp, like with all commands (part of shell not command).

- Can do more things with scp, see tutorial for more: https://www.garron.me/en/articles/scp.html

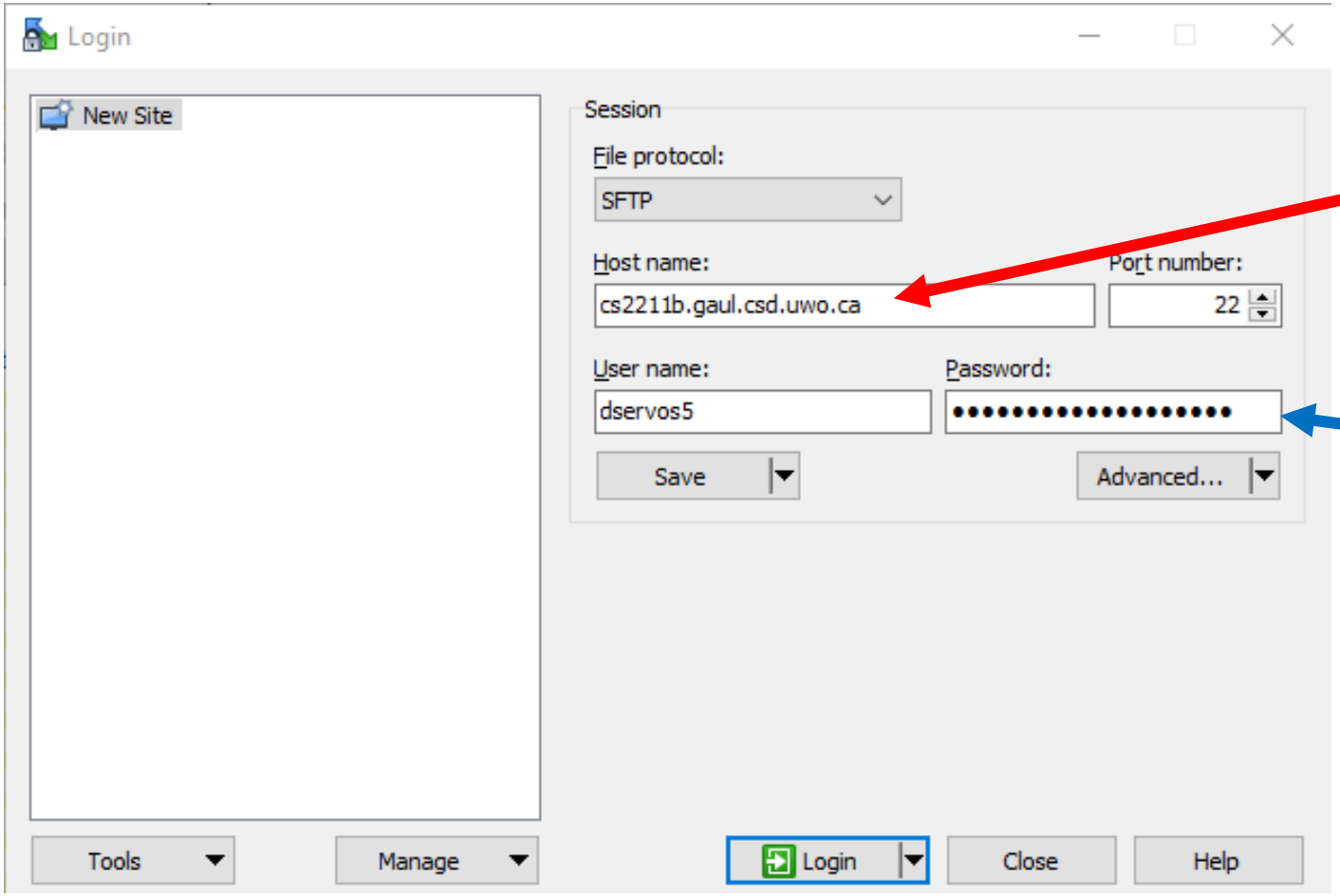
Secure Copy (scp)

With GUI (Windows)

- scp programs available for Windows (not built in).
- Both command line and GUI based.
- Recommended: WinSCP (https://winscp.net)
- Course server also supports SSH File Transfer Protocol (SFTP).
- Many SFTP clients for Windows:
  - WinSCP (also supports SFTP)
  - FileZilla: https://filezilla-project.org/
  - Others.

Secure Copy (scp)

With GUI (Windows)

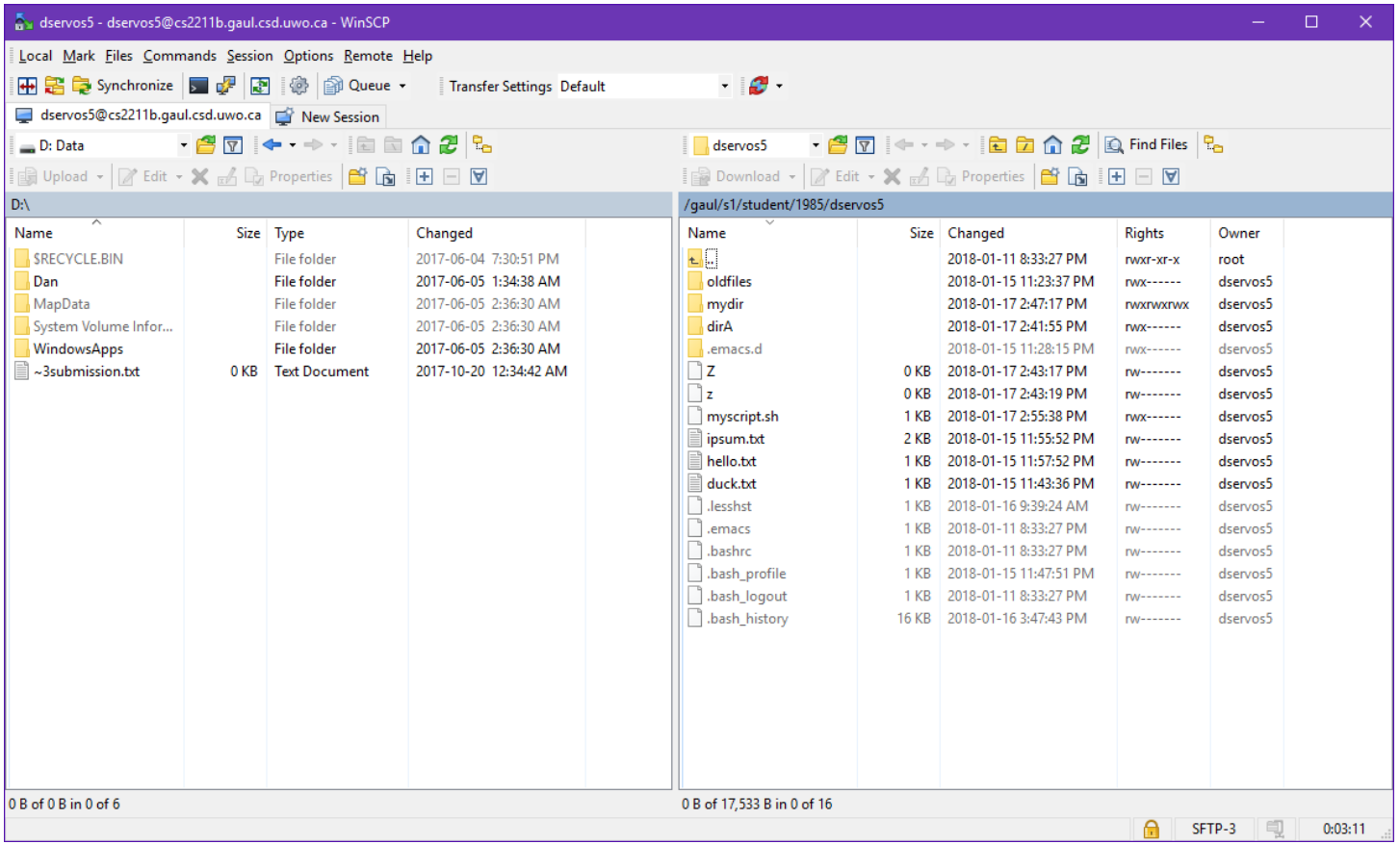


Same hostname

Same GAUL Credentials

Secure Copy (scp)

With GUI (Windows)



Local Files on Left

Remote Files on Right

File Permissions

Access Control Models

- Access control models dictate how access to files, devices and other resources is controlled.
- A number of different models exist:
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Attribute-based Access Control (ABAC)
  - Many others.
- The UNIX file system uses Discretionary Access Control (DAC).

Discretionary Access Control (DAC)

- Every protected object (file, resource, etc.) is assigned a single owner.
- Only the owner can grant permissions on an object they own.
- Permissions are defined as the right to perform some operation (read, write, execute, etc.) on the object.

DAC In UNIX

Owner and Group

- The UNIX file system also adds the notation of user groups to make administration easier.
- Each file is assigned a single owner and group.
- Each user is assigned to one or more groups.
- On the course sever, everyone is assigned to the *gaulusers* group, graduate students are additionally assigned to the *grad* group.
- You can view what groups you or another user are in with the `group` command.

DAC In UNIX

Permissions

- Every file has different permissions for the owner, the group and others (everyone else).
- Permissions are operations the user can do on the file: read, write or execute.
- Owner permissions dictate what the owner of a file may do to it.
- Group permissions apply to everyone in the same user group except the owner of the file.
- Other permissions apply to anyone else not in the same group as the file and not the owner of the file.

Viewing File Permissions

We can view file permissions using `ls -l`:

```
[dservos5@cs2211b ~]$ ls -l
total 16
drwx------ 2 dservos5 grad        33 Jan 17 14:41 dirA
-rwxr-xr-x 1 dservos5 grad       109 Jan 15 23:43 duck.txt
-rw------- 1 dservos5 gaulusers   39 Jan 15 23:57 hello.txt
-rw-rwx--- 1 root     grad      1033 Jan 15 23:55 ipsum.txt
drwxr-xr-x 2 dservos5 gaulusers   33 Jan 17 14:47 mydir
-rwxr-xr-x 1 dservos5 grad        35 Jan 17 14:55 myscript.sh
drwx------ 5 dservos5 grad       153 Jan 15 23:23 oldfiles
```

Viewing File Permissions

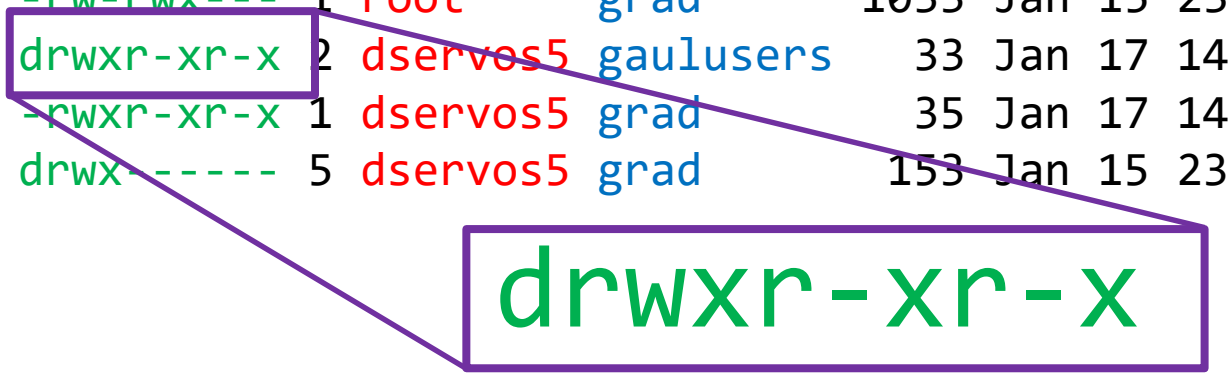
We can view file permissions using `ls -l`:

```
[dservos5@cs2211b ~]$ ls -l
 Permissions     Owner     Group
drwx------ 2 dservos5 grad         33 Jan 17 14:41 dirA
-rwxr-xr-x 1 dservos5 grad        109 Jan 15 23:43 duck.txt
-rw------- 1 dservos5 gaulusers    39 Jan 15 23:57 hello.txt
-rw-rwx--- 1 root     grad       1033 Jan 15 23:55 ipsum.txt
drwxr-xr-x 2 dservos5 gaulusers    33 Jan 17 14:47 mydir
-rwxr-xr-x 1 dservos5 grad         35 Jan 17 14:55 myscript.sh
drwx------ 5 dservos5 grad        153 Jan 15 23:23 oldfiles
```

Viewing File Permissions

We can view file permissions using `ls -l`:

```
[dservos5@cs2211b ~]$ ls -l
 Permissions     Owner     Group
drwx------ 2 dservos5 grad         33 Jan 17 14:41 dirA
-rwxr-xr-x 1 dservos5 grad        109 Jan 15 23:43 duck.txt
-rw------- 1 dservos5 gaulusers    39 Jan 15 23:57 hello.txt
-rw-rwx--- 1 root     grad       1033 Jan 15 23:55 ipsum.txt
drwxr-xr-x 2 dservos5 gaulusers    33 Jan 17 14:47 mydir
-rwxr-xr-x 1 dservos5 grad         35 Jan 17 14:55 myscript.sh
drwx------ 5 dservos5 grad        153 Jan 15 23:23 oldfiles
```

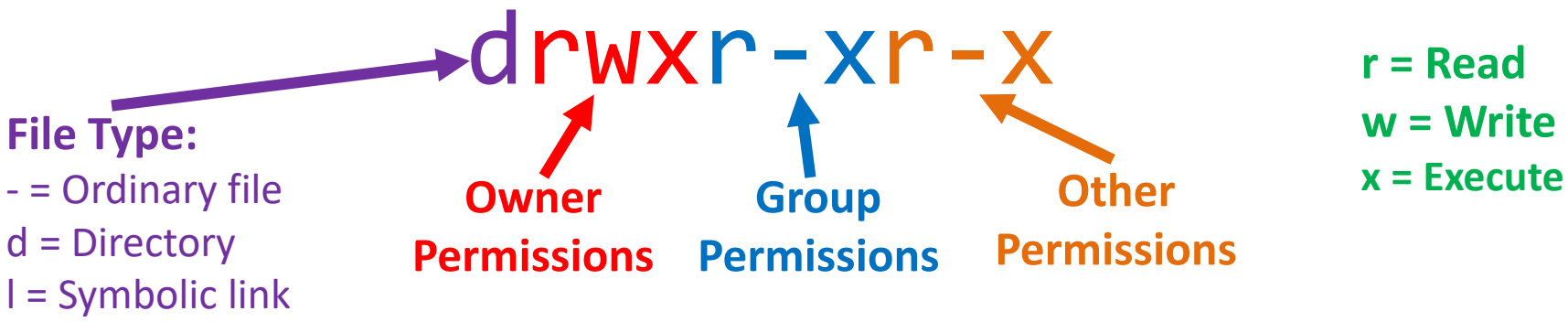


Viewing File Permissions

We can view file permissions using `ls -l`:

```
[dservos5@cs2211b ~]$ ls -l
```

Permissions		Owner	Group				
drwx------	2	dservos5	grad	33	Jan	17 14:41	dirA
-rwxr-xr-x	1	dservos5	grad	109	Jan	15 23:43	duck.txt
-rw-------	1	dservos5	gaulusers	39	Jan	15 23:57	hello.txt
-rw-rwx---	1	root	grad	1033	Jan	15 23:55	ipsum.txt
drwxr-xr-x	2	dservos5	gaulusers	33	Jan	17 14:47	mydir
-rwxr-xr-x	1	dservos5	grad	35	Jan	17 14:55	myscript.sh
drwx------	5	dservos5	grad	153	Jan	15 23:23	oldfiles



Viewing File Permissions

Example 1:

```
[dservos5@cs2211b ~]$ ls -l duck.txt
-rwxr-xr-- 1 dservos5 grad 109 Jan 15 23:43 duck.txt
```

- duck.txt is owned by dservos5
- duck.txt is assigned to the group grad
- duck.txt is an ordinary file
- The owner (dservos5) can read, write and execute duck.txt
- Any one in the grad group can read and execute duck.txt
- Every one else can only read duck.txt

Viewing File Permissions

Example 2:

```
[dservos5@cs2211b ~]$ ls -l ipsum.txt
-rw-rwx--- 1 jdoe32 gaulusers 1033 Jan 15 23:55 ipsum.txt
```

- ipsum.txt is owned by user jdoe32
- ipsum.txt is assigned to the group gaulusers
- ipsum.txt is an ordinary file
- The owner (jdoe32) can read and write ipsum.txt
- Any one in the gaulusers group can read, write, and execute ipsum.txt
- Every one else has no access to ipsum.txt at all

Directory Permissions

Permissions on directories work a bit differently then you might expect:

Operation	Result
Read (r)	Allows user to list the contents of the directory.
Write (w)	Allows a user to delete, move or create a new file in the directory (but only if they also have execute permission).
Execute (x)	Allows a user to “enter” the directory and access the files inside. Can not read/write files without having execute permission on the directory.

Directory Permissions

Examples:

```
d--x------ dir1
```

Owner can read and edit files in dir1 but not delete, move or create new ones. They also can not get a listing of the files in dir1.

```
d-wx------ dir1
```

Owner can read, delete, edit, move and create new files in dir1 but not get a list of files in dir1.

```
dr-------- dir1
```

Owner limited to only listing the files in dir1.

Changing Ownership & Group

chown

- The owner of a file can change the group assigned to the file using the `chown` or `chgrp` command.
- Only the superuser can change the owner of a file using the `chown` command.

```
chown [OPTION]... [OWNER][:[GROUP]] FILE...
```

Changing Ownership & Group

chown

Example 1: Change the owner of *text.txt* to *jdoe32*

```
chown jdoe32 text.txt
```

Example 2: Change the group of *text.txt* to *users*

```
chown :users text.txt
```

Example 3: Change the group of *text.txt* to *grad* and the owner to *dservos5*

```
chown dservos5:grad text.txt
```

Changing File Permissions

Symbolic Method

- The owner of a file can change the granted permissions using the `chmod` command.
- Has a symbolic and octal method of setting permissions.
- Symbolic method:

```
chmod [OPTION]... MODE[,MODE]... FILE..
```

where `MODE` is in the form of:

```
[ugoa][+-=][rwx]
```

Changing File Permissions

Symbolic Method

Examples:

Command	Result
`chmod u=rwx file1`	Set read, write and execute permissions for owner of file1.
`chmod g=wx file1`	Gives the group just write and execute permissions on file1.
`chmod o=r file1`	Gives others only read permissions on file1.
`chmod a=rw file1`	Gives everyone just read and write permissions on file1.
`chmod o= file1`	Remove all permissions for others on file1

u = owner o = others r = read x = execute

g = group a = all w = write

Changing File Permissions

Symbolic Method

Multiple modes can be used together (separated by a comma). Examples:

Command	Result
`chmod u=rwx,g=rw,o= file1`	Set read, write and execute permissions for owner of file1. Just read and write for the group and remove all permissions for others.
`chmod ug=rwx,o=r file1`	Gives owner and group all permissions, and others just read.
`chmod og=rx,u=w file1`	Gives just write to owner, and just read and execute to everyone else.

Changing File Permissions

Symbolic Method

+ and – can be used in place of = to set permissions relative to what they currently are rather than completely replacing them.

Command	Permissions of duck.txt
	-rwxr-xr-x
chmod o-x,g+w duck.txt	
chmod ug-xw,o+w duck.txt	
chmod a-w duck.txt	
chmod u=rw,og+x duck.txt	

How do the permissions of duck.txt change after each command?

Changing File Permissions

Symbolic Method

+ and – can be used in place of = to set permissions relative to what they currently are rather than completely replacing them.

Command	Permissions of duck.txt
	-rwxr-xr-x
chmod o-x,g+w duck.txt	-rwxrwxr--
chmod ug-xw,o+w duck.txt	
chmod a-w duck.txt	
chmod u=rw,og+x duck.txt	

How do the permissions of duck.txt change after each command?

Changing File Permissions

Symbolic Method

+ and – can be used in place of = to set permissions relative to what they currently are rather than completely replacing them.

Command	Permissions of duck.txt
	-rwxr-xr-x
chmod o-x,g+w duck.txt	-rwxrwxr--
chmod ug-xw,o+w duck.txt	-r--r--rw-
chmod a-w duck.txt	
chmod u=rw,og+x duck.txt	

How do the permissions of duck.txt change after each command?

Changing File Permissions

Symbolic Method

+ and – can be used in place of = to set permissions relative to what they currently are rather than completely replacing them.

Command	Permissions of duck.txt
	-rwxr-xr-x
chmod o-x,g+w duck.txt	-rwxrwxr--
chmod ug-xw,o+w duck.txt	-r--r--rw-
chmod a-w duck.txt	-r--r--r--
chmod u=rw,og+x duck.txt	

How do the permissions of duck.txt change after each command?

Changing File Permissions

Symbolic Method

+ and – can be used in place of = to set permissions relative to what they currently are rather than completely replacing them.

Command	Permissions of duck.txt
	-rwxr-xr-x
chmod o-x,g+w duck.txt	-rwxrwxr--
chmod ug-xw,o+w duck.txt	-r--r--rw-
chmod a-w duck.txt	-r--r--r--
chmod u=rw,og+x duck.txt	-rw-r-xr-x

How do the permissions of duck.txt change after each command?

Changing File Permissions

Octal Method

- We can also set the value of permissions using octal values.
- Octal method:

```
chmod [OPTION]… DDD FILE…
```

where DDD are 3 octal digits whose bits represent permissions

Can think of rwx rwx rwx as 111 111 111 in binary
= 7 7 7

`chmod 777 file` would grant full permissions to everyone

Changing File Permissions

Octal Method

#	Permission	rwx
7	read, write and execute	rwx
6	read and write	rw-
5	read and execute	r-x
4	read only	r--
3	write and execute	-wx
2	write only	-w-
1	execute only	--x
0	none	---

Default Permissions

umask

- The `umask` command sets the default permissions for new files you create.
- Takes an octal mode that sets what permissions WILL NOT be set.
- Get resulting permissions by subtracting umask value from 666 for files and 777 for directories.

Example: `umask 022`

```
New Files: 666-022 = 644  i.e. rw- r-- r--
New Dirs:  777-022 = 755  i.e. rwx r-x r-x
```

Links

More on inodes

- Every file associated with a data structure called an inode.
- Stores metadata about a file including:
  - File type (regular, directory, link, device, etc.)
  - File Permissions
  - Number of hard links
  - Owner
  - Group
  - File size
  - Last modification time
  - Last access time
  - Pointers to disk blocks on the hard drive (location of file contents)
- Directory files store the file name and an inode number for each regular file or subdirectory they contain

Hard Links

- Hard links create a new entry in the directory for the file.
- Can be created with the `ln` command.

```
ln TARGET LINK_NAME
```

Hard Links

- Hard links create a new entry in the directory for the file.
- Can be created with the `ln` command.

```
ln TARGET LINK_NAME
```

Hard Links

Example: `ln myfile.txt link.txt`

Before

Directory: dir1	
Filename	**Inode Number**
.	384555
..	453345
myfile.txt	593234
somedir	532453

`ln myfile.txt link.txt`

After

Directory: dir1	
Filename	**Inode Number**
.	384555
..	453345
myfile.txt	593234
somedir	532453
link.txt	593234

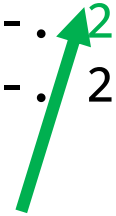
Directory dir1 contains the file *myfile.txt* and the subdirectory *somedir*.

link.txt is added to dir with the same inode number as myfile.txt

Hard Links

Example: `ln myfile.txt link.txt`

```
[dservos5@cs2211b dir1]$ ln myfile.txt link.txt
[dservos5@cs2211b dir1]$ ls -l
total 8
-rw-------. 2 dservos5 grad 31 Jan 17 23:47 link.txt
-rw-------. 2 dservos5 grad 31 Jan 17 23:47 myfile.txt
drwx------. 2 dservos5 grad  6 Jan 17 23:47 somedir
```



Link count is increase to 2

Hard Links

Example: `rm myfile.txt`

Before

Directory: dir1	
Filename	**Inode Number**
.	384555
..	453345
myfile.txt	593234
somedir	532453
link.txt	593234

Directory dir1 with two links to inode 593234.

After

Directory: dir1	
Filename	**Inode Number**
.	384555
..	453345
somedir	532453
link.txt	593234

Can still access file via link.txt. Only entry from dir1 is removed, not file contents.

Hard Links

Example: `rm myfile.txt`

Link count is decreased to 1

```
[dservos5@cs2211b dir1]$ rm myfile.txt
[dservos5@cs2211b dir1]$ ls -l
total 4
-rw-------. 1 dservos5 grad 31 Jan 17 23:47 link.txt
drwx------. 2 dservos5 grad  6 Jan 17 23:47 somedir
[dservos5@cs2211b dir1]$ cat link.txt
This is the text in myfile.txt
```

Can still access file.

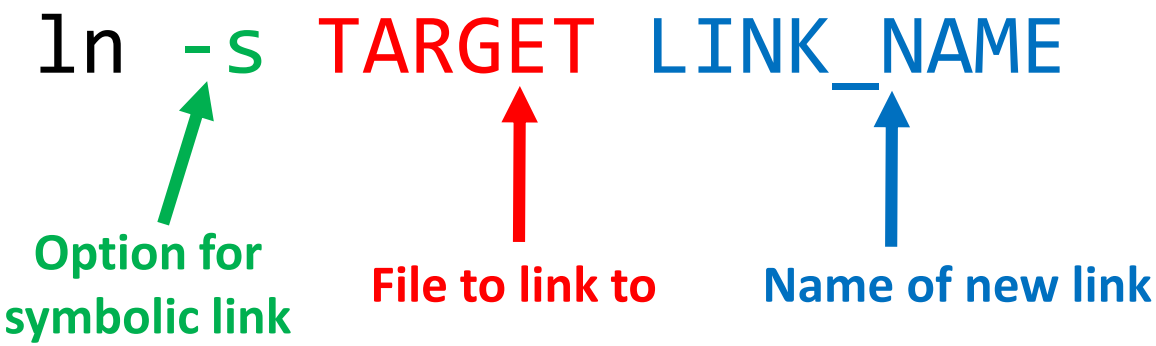
Symbolic (Soft) Links

- Symbolic (soft) links create a new file with a path name that leads to the file it points to.
- Can also be created with the ln command.

```
ln -s TARGET LINK_NAME
```

Symbolic (Soft) Links

- Symbolic (soft) links create a new file with a path name that leads to the file it points to.
- Can also be created with the ln command.



Symbolic (Soft) Links

Example: `ln -s myfile.txt link.txt`

Before

Directory: dir1	
Filename	**Inode Number**
.	384555
..	453345
myfile.txt	593234
somedir	532453

`ln -s myfile.txt link.txt`

After

Directory: dir1	
Filename	**Inode Number**
.	384555
..	453345
myfile.txt	593234
somedir	532453
link.txt	642345

Directory dir1 contains the file *myfile.txt* and the subdirectory *somedir*.

link.txt is added to dir but is a link file (not regular) and a different inode number.

Symbolic (Soft) Links

Example: `ln -s myfile.txt link.txt`

```
[dservos5@cs2211b dir1]$ ln -s myfile.txt link.txt
[dservos5@cs2211b dir1]$ ls -l
total 4
lrwxrwxrwx. 1 dservos5 grad 10 Jan 18 00:02 link.txt -> myfile.txt
-rw-------. 1 dservos5 grad 31 Jan 17 23:47 myfile.txt
drwx------. 2 dservos5 grad  6 Jan 17 23:47 somedir
```

Did not increase the link count

Shows link in ls -l

Symbolic (Soft) Links

Example: `rm myfile.txt`

```
[dservos5@cs2211b dir1]$ rm myfile.txt
[dservos5@cs2211b dir1]$ cat link.txt
cat: link.txt: No such file or directory
```

- Link is now broken.
- File has been deleted.