Debian Administration

# Session 4

Saturday, December 14, 2013

## Part I: Using Regular Expressions

*Candidates should be able to manipulate files and text data using regular expressions. This objective includes creating simple regular expressions containing several notational elements. It also includes using regular expression tools to perform searches through a filesystem or file content.*

### Key Knowledge Areas

* Create simple regular expressions containing several notational elements.
* Use regular expression tools to perform searches through a filesystem or file content.

***Overview***

Finding a word or multiple words in a text is achieved using **grep**, **fgrep** or **egrep**. The keywords used during a search are a combination of letters called *regular expressions.* Regular expressions are recognised by many other applications such as **sed**, and **vi**.

**Regular Expressions**

Traditional Regular Expressions (regex)

A regular expression is a sequence of characters (or atoms) used to match a pattern. Characters are either constants (treated literally) or metacharacters.

*Table1: Main metacharacters*

|  |  |
| --- | --- |
| ***Characters*** | ***Search Match*** |
| \<KEY | Words beginning with ‘KEY’ |
| WORD**\>** | Words ending with ‘WORD’ |
| **^** | Beginning of a line |
| **$** | End of a line |
| **[** Range **]** | Range of ASCII characters enclosed |
| **[^**c **]** | Not the character ‘c’ |
| **\**[ | Interpret character ‘[‘ literally |
| “ca\*t” | Strings containing ‘c’ followed by no 'a' or any number of the letter 'a' followed by a 't' |
| “.” | Match any single character |

**Extended regex**:

The main eregex’s are: +,?,() and |

*Table2: List of main eregex*

|  |  |
| --- | --- |
| ***Characters*** | ***Search Match*** |
| "A1|A2|A3" | Strings containing ‘A1’ or ‘A2’ or ‘A3’ |
| "ca+t" | Strings containing a 'ca' followed by any number of the letter 'a' followed by a 't' |
| "ca?t" | Strings containing ‘c’ followed by no 'a' or exactly one 'a' followed by a 't' |

***The grep family***

The greputility supports regular expressions *regex* such as those listed in *Table1*.

**Working with basic grep**

*Syntax for grep:*

***grep PATTERN FILE***

*Options for grep include:*

|  |  |
| --- | --- |
| ***grep*** | ***Main Options*** |
| **-c** | count the number of lines matching PATTERN |
| **-f** | obtain PATTERN from a file |
| **-i** | ignore case sensitivity |
| **-n** | Include the line number of matching lines |
| **-v** | output all lines except those containing PATTERN |
| **-w** | Select lines only if the pattern matches a whole word. |

For example list all non blank lines in /etc/lilo.conf:

|  |
| --- |
| $ grep –v “^$” /etc/lilo.conf |

**egrep**

The egrep tool supports extended regular expressions *eregex* such as those listed in *Table2*.

The egrep utility will handle any modern regular expressions. It can also search for several keywords if they are entered at the command line, separated by the vertical bar character.

For example;

|  |
| --- |
| $ egrep 'linux|^image' /etc/lilo.conf |

**fgrep**

fgrep stands for *fast grep* and fgrep interprets strings literally (no regex or eregex support). The **fgrep** utility does not recognise the special meaning of the regular expressions.

For example

|  |
| --- |
| $ fgrep 'cat\*' *FILE* |

will only match words containing ‘cat\*’. The main improvement came from **fgrep**’s ability to search from a list of keywords entered line by line in a file, say LIST. The syntax would be

|  |
| --- |
| $ fgrep –f LIST FILE |

The Stream Editor - sed

**sed** performs automatic, non-interactive editing of files. It is often used in scripts to search and replace patterns in text. It supports most regular expressions.

Syntax for sed:

**sed [options] 'command' [INPUTFILE]**

The input file is optional since **sed** also works on file redirections and pipes. Here are a few examples assuming we are working on a file called MODIF.

*Delete all commented lines:*

|  |
| --- |
| $ sed '/^#/ d ' *MODIF* |

Notice that the search pattern is between the double slashes.

*Substitute /dev/hda1 by /dev/sdb3:*

|  |
| --- |
| $ sed 's**/**\/dev\/hda1**/**\/dev\/sdb3**/**g' *MODIF* |

The **s** in the command stands for ‘substitute’. The **g** stands for “globally” and forces the substitution to take place throughout each line. You can also specify which line numbers the substitutions should occur on, either using line numbers or regular expression match.

*If the line contains the keyword KEY then substitute ‘:’ with ‘;’ globally:*

|  |
| --- |
| $ sed '/KEY/ s/:/;/g' *MODIF* |

More Advanced sed

You can issue **several commands** each starting with **–e** at the command line. For example, (1) delete all blank lines then (2) substitute ‘OLD’ by ‘NEW’ in the file *MODIF*

|  |
| --- |
| $ sed –e '/^$/ d’ **-e** ‘s/OLD/NEW/g' MODIF |

These commands can also be written to a file, say COMMANDS. Then each line is interpreted as a new command to execute (no quotes are needed).

|  |
| --- |
| ***An example COMMANDS file*** |
| 1 s/old/new/ |
| /keyword/ s/old/new/g |
| 23,25 d |

The syntax to use this *COMMANDS* file is:

**sed -f COMMANDS MODIF**

This is much more compact than a very long command line !

*Summary of options for* ***sed***

|  |
| --- |
| ***Command line flags*** |
| **-e** Execute the following command |
| **-f** Read commands from a file |
| **-n** Do not printout unedited lines |

|  |
| --- |
| ***sed commands*** |
| **d** Delete an entire line |
| **r** Read a file and append to output |
| **s** Substitute |
| **w** Write output to a file |

Used files, terms and utilities:

* grep
* egrep
* fgrep
* sed
* regex(7)

## Part II: Maintain the integrity of file systems

*Candidates should be able to maintain a standard filesystem, as well as the extra data associated with a journaling filesystem.*

### Key Knowledge Areas

* Verify the integrity of filesystems.
* Monitor free space and inodes.
* Repair simple filesystem problems.

Once a filesystem has been created on your block device you will want to know how to monitor the filesystem and check it for errors, recovering from errors where possible. Fortunately the filesystem provide several command and utilities to aid in this process.

***Monitoring Disk Usage***

The **df** (disk free) and disk usage (**du**) commands can be used to report on the amount of free disk space and query how much space directories and files are using.***df*** works on a device level, as opposed to a directory level.

The **df** tool shows used and available disk space. By default this is given in blocks of 1K.

|  |
| --- |
| $ df -h  Filesystem Size Used Avail Use% Mounted on  /dev/hda9 289M 254M 20M 93% /  /dev/hda2 23M 7.5M 14M 35% /boot  none 62M 0 61M 0% /dev/shm  /dev/hda5 1.4G 181M 1.1G 13% /share  /dev/hda7 787M 79M 669M 11% /tmp  /dev/hda3 4.3G 3.4G 813M 81% /usr  /dev/hda6 787M 121M 627M 17% /var  //192.168.123.2/share 12G 8.8G 3.7G 71% /mnt/smb |

The du command will display disk usage. This is done on a per directory basis. du cannot display available space since this information is only available at a device level.

The following command will list the current usage of the **/etc** directory in human readable units (using the **-h** switch) and will only print the grand total (using the **-s** switch):

|  |
| --- |
| *# du -sh /etc*  *62M /etc/* |

***File System Checking, Repair and Maintenance***

As with filesystem monitoring there are numerous tools for maintaining a Linux filesystem which are provided by the creators of the respective filesystem.

If the file system is damaged or corrupt, then the fsck utility should be run against the partition (the minimum requirement is that the file system be unmounted or mounted read-only).

**fsck** acts as a front that automatically detects the file system type of a partition. Then as with **mkfs**, the tools fsck.ext2, **fsck.ext3** or **fsck.xfs** will be called accordingly to carry out the system check and, if necessary, repair. Since ext3 is the main filesystem type for Linux there is a e2fsck command that only handles this filesystem type.

You invoke the filesystem check as follows explicitly specify a file system type with the following syntax:

fsck –t <fstype> <device>

Example: Checking a *reiserfs* filesystem on the /dev/sdb10 device:

|  |
| --- |
| # fsck –t reiserfs /dev/sdb10  # fsck.reiserfs /dev/sdb10 |

**Ext File System Maintenance utilities**

As the extended filesystem is the most widely used and deployed filesystem on Linux, the tools for ext filesystem support are more numerous and comprehensive than for other filesystems.

***Ext File System Debug Commands***

The debugfs and dumpe2fs are seldom used but can be useful in providing low level information about an ext2 or ext3 filesystem.

*debugfs [ -b blocksize ] [ -s superblock ] [ -f cmd\_file ] [ -R request ] [ -V ] [ [ -w ] [ -c ] [ -i ] [ device ] ]*

The ***debugfs*** program is an interactive file system debugger. It can be used to examine and change the state of an ext2/3 file system.

Once in the ***debugfs*** shell, internal commands can then be used to change directory, examine inode data, remove files, create links, dump the ext3 journal logs etc. While this is a very powerful command, it should be used with caution, generally only after the fsck command has failed to make any headway.

***dumpe2fs*** *[ -bfhixV ] [ -ob superblock ] [ -oB blocksize ] device*

dumpe2fs prints the super block and block group information for the filesystem present on *device*.

|  |
| --- |
| dumpe2fs /dev/hda1  dumpe2fs 1.35 (28-Feb-2004)  Filesystem volume name: /boot1  Last mounted on: <not available>  Filesystem UUID: d741042c-3eaf-49ee-94c1-7dd8c5459764  Filesystem magic number: 0xEF53  Filesystem revision #: 1 (dynamic)  Filesystem features: has\_journal ext\_attr resize\_inode dir\_index filetype needs\_recovery sparse\_super  Default mount options: (none)  Filesystem state: clean  Errors behavior: Continue  Filesystem OS type: Linux  Inode count: 25584  Block count: 102280  Reserved block count: 5114  Free blocks: 80564  Free inodes: 25537  First block: 1  Block size: 1024  Fragment size: 1024  Reserved GDT blocks: 256  Blocks per group: 8192  Fragments per group: 8192  Inodes per group: 1968  Inode blocks per group: 246  Filesystem created: Sat May 7 10:40:51 2005  Last mount time: Sun May 29 04:08:01 2005  Last write time: Sun May 29 04:08:01 2005  Mount count: 10  Maximum mount count: -1  Last checked: Sat May 7 10:40:51 2005  Check interval: 0 (<none>)  Reserved blocks uid: 0 (user root)  Reserved blocks gid: 0 (group root)  First inode: 11  Inode size: 128  Journal inode: 8  Default directory hash: tea  Directory Hash Seed: 50108791-6a0a-41ff-9608-0485c993eaf9  Journal backup: inode blocks  Group 0: (Blocks 1-8192)  Primary superblock at 1, Group descriptors at 2-2  Block bitmap at 259 (+258), Inode bitmap at 260 (+259)  Inode table at 261-506 (+260)  0 free blocks, 1942 free inodes, 2 directories  Free blocks:  Free inodes: 27-1968  [....] |

**tune2fs**

*tune2fs* allows you to adjust various filesystem parameters on Linux extended filesystems. The following is a list of the most common parameters used to adjust extended filesystem settings:

* -c sets the number of times a filesystem will be mounted before a filesystem check is forced. This is usually at next boot but can be run when a filesystem is manually unmounted.
* -C sets the number of times the filesystem was mounted since it was last checked.
* -L sets the volume label, this used to be used to uniquely identify hard disk partitions but is being replaced with UUIDs.
* -i sets the maximum time between filesystem checks. A filesystem check will be forced when either the time expires or the the maximum number of mounts has been exceeded, which ever comes first.
* -j adds journaling to an ext2 filesysem making it an ext3 filesystem.

Running tune2fs -l will print out the current settings for a filesystem.

|  |
| --- |
| tune2fs 1.41.11 (14-Mar-2010)  Filesystem volume name: <none>  Last mounted on: /  Filesystem UUID: 6f4746c6-777e-4937-92ee-de98cf8f5aa4  Filesystem magic number: 0xEF53  Filesystem revision #: 1 (dynamic)  Filesystem features: has\_journal ext\_attr resize\_inode dir\_index filetype needs\_recovery extent flex\_bg sparse\_super large\_file huge\_file uninit\_bg dir\_nlink extra\_isize  Filesystem flags: signed\_directory\_hash  Default mount options: (none)  Filesystem state: clean  Errors behavior: Continue  Filesystem OS type: Linux  Inode count: 26558464  Block count: 106205707  Reserved block count: 5310285  Free blocks: 62912050  Free inodes: 25721258  First block: 0  Block size: 4096  Fragment size: 4096  Reserved GDT blocks: 998  Blocks per group: 32768  Fragments per group: 32768  Inodes per group: 8192  Inode blocks per group: 512  RAID stride: 32747  Flex block group size: 16  Filesystem created: Sat Feb 13 00:48:21 2010  Last mount time: Tue Sep 28 07:27:39 2010  Last write time: Tue Sep 21 09:14:03 2010  Mount count: 22  Maximum mount count: 25  Last checked: Tue Sep 21 09:14:03 2010  Check interval: 15552000 (6 months)  Next check after: Sun Mar 20 09:14:03 2011  Lifetime writes: 709 GB  Reserved blocks uid: 0 (user root)  Reserved blocks gid: 0 (group root)  First inode: 11  Inode size: 256  Required extra isize: 28  Desired extra isize: 28  Journal inode: 8  First orphan inode: 404853  Default directory hash: half\_md4  Directory Hash Seed: 982f8e6c-db49-49b0-8f65-bce3725b5196  Journal backup: inode blocks |

XFS File System Maintenance Utilities

The tools that come with XFS for filesystem integrity checking are xfs\_info and xfs\_metadump.

xfs\_metdump is a filesystem debugging utility, that dumps xfs filesystem meta-data to a file. The file can then be used to debug the files or as a backup. Later the meta-data can be restored with the xfs\_restore utility.

Used files, terms and utilities:

* du
* df
* fsck
* e2fsck
* mke2fs
* debugfs
* dumpe2fs
* tune2fs
* xfs tools (such as xfs\_metadump and xfs\_info)

## Part III: Manage File Permissions and Ownership

*Candidates should be able to control file access through the proper use of permissions and ownerships.*

### Key Knowledge Areas

* Manage access permissions on regular and special files as well as directories.
* Use access modes such as suid, sgid and the sticky bit to maintain security.
* Know how to change the file creation mask.
* Use the group field to grant file access to group members.

**File and Directory Permissions**

Access to directories and files on Linux is controlled by a simple file permissions systems. Every file/directory has permissions for the file owner, the group to which the file belongs and other, that is users who are not the owner and do not belong to the group to which the file belongs. The permissions are known as the access mode of file/directory and can be viewed by running the **ls -l** command.

File access modes are displayed symbolically as group of 3 letters or numerically as a set of 3 octal digits, but represent a 9 bit number, with each bit representing an access right.

|  |
| --- |
| drwxr-xr-x 3 root root 4.0K 2009-10-27 20:03 hal  -rw-r--r-- 1 root root 4.7K 2009-10-06 22:45 hdparm.conf  -rw-r--r-- 1 root root 92 2009-04-27 11:56 host.conf  -rw-r--r-- 1 root root 4 2010-02-13 01:03 hostname  -rw-r--r-- 1 root root 292 2010-06-24 11:57 hosts  -rw-r--r-- 1 root root 579 2009-10-27 20:12 hosts.allow |

The extract above is from running the ls -l command on the

/etc directory. When a file is created it is owned by the user who created the file and assigned to the default group of the owner.

**Symbolic and Octal Notation**

Permissions can be read=r, write=w and execute=x. The octal values of these permissions are listed in the next table.

*Octal and symbolic permissions*.

|  |  |  |
| --- | --- | --- |
| ***Symbolic*** | ***Octal*** | ***Binary*** |
| read | 4 | ' 100' |
| write | 2 | ' 010' |
| execute | 1 | ' 001' |

Permissions apply to the user, the group and to others. An item has a set of 3 grouped permissions for each of these categories.

*How to read a 755 or -rwxr-xr-x permission*

|  |  |  |
| --- | --- | --- |
| user | group | other |
| rwx  4+2+1=**7** | r-x  4+1=**5** | r-x  4+1=**5** |

**The Standard Permissions & UMASK**

UNIX system create files and directories with standard permissions as follows:

Standard permission for:

Files 666 -rw-rw-rw-

Directories 777 -rwxrwxrwx

Every user has a defined umask that alters the standard permissions. The **umask** applies only at the point at which the file is created. The **umask** has an octal value and is subtracted(\*) from the octal *standard permissions* to give the file's permission (this permission doesn't have a name and could be called the file's *effective* permission).

(\*) While subtraction works in most cases, it should be noted that technically the standard permissions and the umask are combined as follows:

|  |
| --- |
| Final Permissions = Standard Permissions (logical AND) (NOT) umask |

On systems where users belong to separate groups, the umask can have a value of 002.

For systems which place all users in the *users* group, the umask is likely to be 022 so that files do not have group write access by default.

**Octal umask Mode 022 And 002**

If the default settings are not changed, files are created with the access mode 666 and directories with 777. In this example:

1. The default **umask 002** used for normal user. With this mask default directory permissions are 775 and default file permissions are 664.
2. The default **umask for the root user is 022** result into default directory permissions are 755 and default file permissions are 644.
3. For directories, the **base permissions** are (rwxrwxrwx) 0777 and for files they are 0666 (rw-rw-rw).

In short,

1. A umask of **022** allows only you to write data, but anyone can read data.
2. A umask of **077** is good for a completely private system. No other user can read or write your data if umask is set to 077.
3. A umask of **002** is good when you share data with other users in the same group. Members of your group can create and modify data files; those outside your group can read data file, but cannot modify it. Set your umask to **007** to completely exclude users who are not group members.

**How Do I Calculate umasks?**

The octal umasks are calculated via the bitwise AND of the unary complement of the argument using bitwise NOT. The octal notations are as follows:

**Octal value** : Permission

**0** : read, write and execute

**1** : read and write

**2** : read and execute

**3** : read only

**4** : write and execute

**5** : write only

**6** : execute only

**7** : no permissions

Now, you can use above table to calculate file permission. For example, if umask is set to 077, the permission can be calculated as follows:

|  |  |  |
| --- | --- | --- |
| **Bit** | **Targeted at** | **File permission** |
| 0 | Owner | read, write and execute |
| 7 | Group | No permissions |
| 7 | Others | No permissions |

To set the umask 077 type the following command at shell prompt:

$ umask 077  
$ mkdir dir1  
$ touch file  
$ ls -ld dir1 file

**Task: Calculating The Final Permission For FILES**

You can simply subtract the umask from the base permissions to determine the final permission for file as follows:

666 - 022 = 644

* File base permissions : 666
* umask value : 022
* subtract to get permissions of new file (666-022) : 644 (rw-r--r--)

**Task: Calculating The Final Permission For DIRECTORIES**

You can simply subtract the umask from the base permissions to determine the final permission for directory as follows:

777 - 022 = 755

* Directory base permissions : 777
* umask value : 022
* Subtract to get permissions of new directory (777-022) : 755 (rwxr-xr-x)

How Do I Set umask Using Symbolic Values?

The following symbolic values are used:

1. **r**: read
2. **w** : write
3. **x** : execute
4. **u** : User ownership (user who owns the file)
5. **g** : group ownership (the permissions granted to other users who are members of the file's group)
6. **o** : other ownership (the permissions granted to users that are in neither of the two preceding categories)

The following command will set umask to 077 i.e. a umask set to u=rwx,g=,o= will result in new files having the modes -rw-------, and new directories having the modes drwx------:

$ umask u=rwx,g=,o=  
$ mkdir dir2  
$ touch file2  
$ ls -ld dir2 file2

Sample umask Values and File Creation Permissions

|  |  |  |  |
| --- | --- | --- | --- |
| **umask value** | **User permission** | **Group permission** | **Others permission** |
| 000 | all | all | all |
| 007 | all | all | none |
| 027 | all | read / execute | none |

all = read, write and executable file permission

**Limitations of the umask**

1. The umask command can restricts permissions.
2. The umask command cannot grant extra permissions beyond what is specified by the program that creates the file or directory. If you need to make permission changes to existing file use the chmod command.

**umask and level of security**

The umask command be used for setting different security levels as follows:

|  |  |  |
| --- | --- | --- |
| **umask value** | **Security level** | **Effective permission (directory)** |
| 022 | Permissive | 755 |
| 026 | Moderate | 751 |
| 027 | Moderate | 750 |
| 077 | Severe | 700 |

***Changing permissions and owners***

From the previous figure we see that permissions can be acted upon with chmod. There are 3 categories of ownership for each file and directory:

**u**: user

**g**: group

**o**: other

Example:

-rw-rw-r-- 1 jade sales 24880 Oct 25 17:28 libcgic.a

Changing Permissions with **chmod**:

|  |
| --- |
| #chmod g=r,o-r libcgic.a  #chmod g+w libcgic.a |

Changing user and group with chown and chgrp :

|  |
| --- |
| #chown root libcgic.a  #chgrp apache libcgic.a |

***NOTE***:

A useful option for **chmod**, **chown** and **chgrp** is **–R** which recursively changes ownership and permissions through all files and directories indicated.

Special Permissions

**SUID Permissions**

An executable can be assigned a special permission which will always make it run as the owner of this file. This permission is called SUID meaning 'set user ID'. It has a symbolic value **s** or a numerical value **4000**.

Administrative tools may have the SUID bit set in order to allow non-root users to change system files.

For example the **passwd** command can be run by any user and will interactively change his or her current password. This password will be saved to **/etc/shadow**. However this file belongs to user root with typical permissions of 600.

This problem has been solved by setting the SUID bit on **passwd** hence forcing it to run as user root with the correct permissions to modify **/etc/shadow**.

The SUID on **passwd**

|  |
| --- |
| # ls -l $(which passwd)  -r-s--x--x 1 root root 18992 Jun 6 2003 /usr/bin/passwd |

**NOTE:**

The SUID bit is shown in symbolic form in the command above. It is possible to get more information about a file using **stat** as well as seeing the octal representation of the permissions as follows:

|  |
| --- |
| # stat /usr/bin/passwd  File: '/usr/bin/passwd'  Size: 18992 Blocks: 40 IO Block: 4096 regular file  Device: 305h/773d Inode: 356680 Links: 1  Access: (4511/-r-s--x--x) Uid: ( 0/ root) Gid: ( 0/ root) |

|  |
| --- |
| WARNING! WARNING! WARNING! |
| The SUID permission is often associated with security issues. Here is an example that illustrates this. |
| 1. A user would like to read user root's mail. For this he changes the MAIL environment variable as follows: |
| # export MAIL=/var/spool/mail/root |
| 2. The user then uses the command **mail**, hoping to see something! |
| # mail  /var/spool/mail/root: Permission denied |
| So far it doesn't work. This would be too easy! |
|  |
| But if root can be convinced to set the SUID bit on **mail** the previous commands would allow any user to read anybody's mail (including root). |

The next examples are dangerous. Why?

|  |
| --- |
| #chmod 4755 /bin/cat  #chmod u+s /bin/grep |

***SGID permissions***

The SGID is permission similar to SUID that is set for group members. The symbolic value is **s** and the octal value of **2000**.

Setting SGID on a directory changes the group ownership used for files subsequently created in that directory to the directory's group ownership. No need to use newgrp to change the effective group of the process prior to file creation.

Examples:

|  |
| --- |
| #chmod 2755 /home/data  #chmod g+s /bin/wc |

**The sticky bit**

The sticky bit permissionwith value **1000** has the following effect:

* Applied to a directory it prevents users from deleting files unless they are the owner (ideal for directories shared by a group, or for **/tmp**
* Applied to a file this used to cause the file or executable to be loaded into memory and caused later access or execution to be faster. The symbolic value for an executable file is **t** . It was supported in some versions of Unix but is not used in Linux.

Examples:

|  |
| --- |
| #chmod 1666 /data/store.txt  #chmod o+t /home/students |

Used files, terms and utilities:

* chmod
* umask
* chown
* chgrp

## Part IV: Create and change hard and symbolic links

*Candidates should be able to create and manage hard and symbolic links to a file*

### Key Knowledge Areas

* Create links.
* Identify hard and/or softlinks.
* Copying versus linking files.
* Use links to support system administration tasks.

When we copy a file with the cp command, a duplicate of that file is created, sometimes however we want to provide an link to an existing file but want that path to point to the exact same file as the original. In this case we will use symbolic links.

**Symbolic Links**

A soft link to a file or a directory is a special file type that simply contains the name of the file that it “points to”.

|  |
| --- |
| # ln -s mytext.txt myext.sym |

Soft links can be created across filesystems. By running **ls -l** we can identify whether a directory entry is a soft link or just an ordinary file from the output. A symbolic link is shown as follows when we run the **ls -l** command

mytext.txt -> mytext.sym. Notice that the reference count is **1** for both files.

-rw------- **1** root root 223 Sep 29 09:10 mytext.txt

lrwxrwxrwx **1** root root 9 Sep 29 09:10 mytext.sym -> mytext.txt

To find all symbolic links to a file you can use the find command for example

find / -lname mytext.txt

will find all symbolic links to the file mytext.txt.

**Hard Links**

A hard link is an additional name for the same inode and as such the reference count of the file increases by one for every new hard link.

|  |
| --- |
| # ln mytextfile.txt mytextfile.link |

In the listing notice that the reference count is **2** and that both files have the same size. In fact they are identical.

-rw------- **2** mark mark 223 Sep 26 09:06 mytextfile.txt

-rw------- **2** mark mark 223 Sep 26 09:06 mytextfile.link

Hard links can only be created within the same filesystem. Using ls, a hard link can be identified by the reference count shown in the output, as in the above example. Another way of finding files with hard links is to obtain the file's inode number and then run the find command with the inode number as a parameter. To find the inode of a file run the command:

|  |
| --- |
| # ls -i mytextfile.txt  8652338 mytextfile.txt |

This will output the inode number of the file, next run the find command as follows:

|  |
| --- |
| # find / -inum 8652338 |

Used files, terms and utilities:

* ln