Unix Filenames

- A file name consists of characters
- Almost *any character* is a valid file name character, including
  - all the punctuation and digits
  - the one exception is the / (slash) character (*the **only** invalid character in a Unix file name*)
  - the following characters are ***not** recommended* to be used in a Unix file name
    - ? * [ ] “ ” ’ ( ) & : ; !
  - the following are ***not** recommended* to be *used as the first character* in a Unix file name
    - - ~
  - control characters are also *allowed*, but are ***not** recommended*
- UPPER and lower case letters are different
  - A.txt and a.txt are different files

Unix Filenames

- Unix does not enforce file name extensions
  - The following are all legal Unix file names
    - a
    - a.
    - .a
    - …
    - a.b.c
- Remember any name beginnings with a dot is hidden
  - ls cannot see them, but ls -a can
- . and .. are reserved for current and parent directories

Unix Filenames

- Executable files usually have no extensions
- Distinguishing executable files from data files might be difficult
- "file" command can help in this regard

  Use: file filename

  Result: print the type of the file

  Example: obelix[1] > file ~/.cshrc
  .cshrc: executable c-shell script
- Filenames and pathnames have limits on lengths
  - Typically, can consist of 255 characters
  - these are pretty long (much better than the late MS-DOS days and the 8.3 filenames)

Fixing Filename Mistakes

- It is very easy to get the wrong stuff into file names
  - Say you accidentally typed

    obelix[3] > cp myfile -i

    Creates a file with a name -i
  - If you attempt to delete it as follow:

    obelix[4] > rm -i
    - The shell thinks -i is an option, not a file
    - Getting rid of these files can be painful
- There is an easy way to fix this...
  - You simply type

    obelix[5] > rm -- -i
  - Many commands use "--" to say there are no more options

Filename Wildcarding

- Wildcarding is the use of "*special*" characters to represent or match a sequence of other characters
  - A short sequence of characters can match
    - a long file name
    - many file names
- Wildcard characters include:

  * matches a sequence of ***zero*** or more characters
    - Example: a*.c* matches for example abc.c, apple.cpp

  ? matches any ***single*** character
    - Example: a?.c matches for example ab.c, ax.c, but *not* abc.c or a.c

  [...] matches any ***single*** character between the braces
    - Example: b[aei]t matches exactly bat, bet, or bit, *not* baet or bt

  Within [...], a pair of characters separated by "-" matches any character lexically between the two
    - Example: [a-z]* matches any file starts by a lower-case letter

Filename Wildcarding

- Wildcard sequences can be combined

  obelix[6] > mv a*.[ch] cfiles/
    - mv all files beginning with a and ending with .c or .h into the directory cfiles

  obelix[7] > ls [abc]*.?
    - list files whose name begins with a, b, or c and ends with . (dot) followed by a single character
- Wildcards do ***not*** cross "/" boundaries
  - Example: cs2211*p ***does not*** match cs2211/tmp
  - Example: cs2211/*p ***does*** match cs2211/tmp
- ***Wildcards are expanded by the shell***
  - ***The program just sees the list of files matched***

Filename Wildcarding

- Matching the dot hidden files
  - A dot (.) at the beginning of a filename must be matched ***explicitly***
  - Example: ls .c*

What does it mean?

  - Note that, ls ?c* will ***not*** match **.c**

Filename Wildcarding

- More advanced examples:
  - What does the following command do?

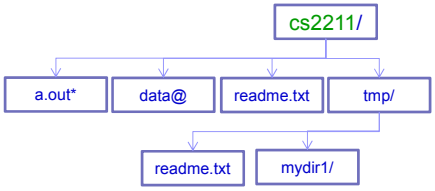
```
obelix[8] > ls -1 /bin/*[-+]*
```

List all non-hidden files or directories in the /bin directory that include at least a "-" or "+" in their name

  - What the differences between

```
obelix[9] > ls
obelix[10] > ls *
obelix[11] > ls */*
```

Remember: Wildcards are expanded by the shell and the program just sees the list of files matched



Filename Wildcarding

- What about this?

  obelix[12] > mv *.bat *.bit

  If "*.bat" and "*.bit" ***match exactly two file names***, then the "mv" command will rename the first file to the second one (the second file will be overridden)

  If both "*.bat" and "*.bit" ***match less than two file names***, you will get an error message saying "mv: Insufficient arguments"

  If "*.bat" and "*.bit" ***match more than two file names and the last one is not a directory***, you will get an error message saying that "mv: Target zzz.bit must be a directory", where zzz.bit is the last file name in the list.

  If "*.bat" and "*.bit" ***match more than two file names and the last one is a directory***, all files will be moved to that directory

Unix Quoting

- Double Quotes: "...."
  - Putting text between double quotes
    - **stops** the interpretation of *some* shell special characters (*whitespaces* and *wildcards*)
    - **forces** the shell to deal with whatever between the two quotes as one token
  - Examples:

    obelix[13] > echo Here are some words
    Here are some words
    obelix[14] > echo "Here are some words"
    Here are some words
    obelix[15] > mkdir "A directory name with spaces?"
    obelix[16] > ls A*
    A directory name with spaces?/

  - ***Question***: what will happen if you execute:
    mkdir A directory name with spaces?

How many arguments do we have in each of the above commands?

Unix Quoting

- Single Quotes '...'
  - Putting text between single quotes **stops** the interpretation of *even more* special characters
    - Stop variable expansion ($HOME, etc.)
    - Examples:

```
obelix[17] > echo "Welcome    $HOME"
Welcome    /faculty/elsakka
obelix[18] > echo 'Welcome    $HOME'
Welcome    $HOME
```


Unix Quoting

- Back Quotes `...`
  - Putting text between back quotes *execute the text as a command and return the results*
    - Note the difference:
      - single quote (')
      - back quote (`)
    - Examples:

```
obelix[19] > cd
obelix[20] > echo "pwd"
pwd
obelix[21] > echo 'pwd'
pwd
obelix[22] > echo `pwd`
/faculty/elsakka
```

Unix Quoting

- Backslash \
  - *quotes* the next character
  - Escapes *all* of the shell special characters

```
obelix[23] > mkdir Dir\ name\ with\ spaces\$\*
obelix[24] > ls Dir\ *
Dir name with spaces$*/
```

  - Use backslash to escape a newline character

```
obelix[25]> echo "This is a long line and\
we want to continue on the next one"
This is a long line and
we want to continue on the next one
```

Must be followed by Enter

  - Use backslash to escape other shell special characters

```
obelix[26] > echo \"This is Mahmoud\'s book\"
"This is Mahmoud's book"
obelix[27] > echo \"This is Mahmoud\'s book\"
"This is Mahmoud's book"
```

How many arguments do we have in each of the above commands?

Unix Quoting

- Exercise *(enjoy it at home ☺)*
- What is the diffidence between the following commands:
  (Try to expect the result before hitting enter)

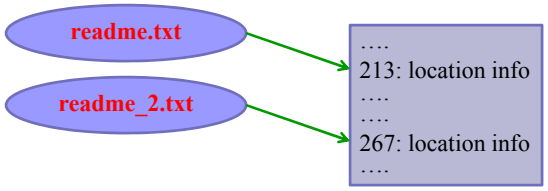
```
echo  ls $HOME
echo "ls $HOME"
echo 'ls $HOME'
echo `ls $HOME`

echo  ls \$HOME
echo "ls \$HOME"
echo 'ls \$HOME'
echo `ls \$HOME`

echo \"ls $HOME\"
echo \'ls $HOME\'
echo \`ls $HOME\`

echo \"ls \$HOME\"
echo \'ls \$HOME\'
echo \`ls \$HOME\`

echo \"ls \$HOME"
echo \'ls \$HOME'
echo \`ls \$HOME`

echo "ls \$HOME\"
echo 'ls \$HOME\'
echo `ls \$HOME\`
```

Hard and Symbolic Links

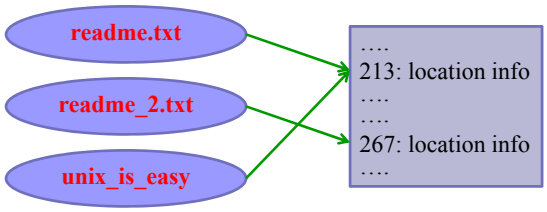
- Unix creates a single *index node* (*inode*) control structure for each file
- When a file is created, there is one link to the file's *inode*
- ls -i display the inode number for each file
- ls -l display the number of existing links per file
- Additional links can be added to a file using the command ln (**hard links**)
- Each hard link acts like a pointer to the actual file

Hard and Symbolic Links

```
obelix[1] > ls -i
213 readme.txt          267 readme_2.txt
obelix[2] > ls -l
-rw-r--r--      1  elsakka  faculty 3456  Sept 10 23:34  readme.txt
-rw-r--r--      1  elsakka  faculty  876  Sept  8 21:02  readme_2.txt
```



Hard and Symbolic Links

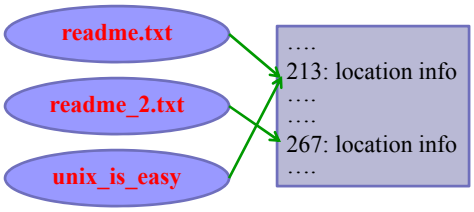


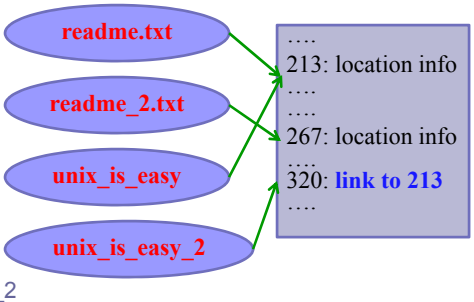

```
obelix[3] > ln readme.txt unix_is_easy
obelix[4] > ls -i
213 readme.txt          267 readme_2.txt  213 unix_is_easy
obelix[5] > ls -l
-rw-r--r--      2  elsakka  faculty 3456  Sept 10 23:34  readme.txt
-rw-r--r--      1  elsakka  faculty   876  Sept   8 21:02  readme_2.txt
-rw-r--r--      2  elsakka  faculty 3456  Sept 10 23:34  unix_is_easy
```

- There is only one copy of the file contents on the hard disk, but has two distinct names!

Hard and Symbolic Links

- A symbolic link "ln -s" is an ***indirect*** pointer to another file or directory

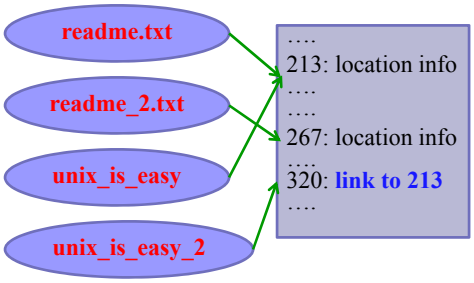





```
obelix[6] > ln -s readme.txt unix_is_easy_2
obelix[4] > ls -i
213 readme.txt          267 readme_2.txt  213 unix_is_easy   320 unix_is_easy_2
obelix[5] > ls -l
-rw-r--r--      2  elsakka  faculty 3456  Sept 10 23:34  readme.txt
-rw-r--r--      1  elsakka  faculty   876  Sept   8 21:02  readme_2.txt
-rw-r--r--      2  elsakka  faculty 3456  Sept 10 23:34  unix_is_easy
lrwxrwxrwx      1  elsakka  faculty     10  Sept 10 23:36  unix_is_easy_2  -> reamme.txt
```

Hard and Symbolic Links

- What will happen if I delete:
  - reame.txt
  - unix_is_easy_2
  - reame.txt and unix_is_easy_2
  - reame.txt and unix_is_easy




```
obelix[4] > ls -i
213 readme.txt          267 readme_2.txt  213 unix_is_easy   320 unix_is_easy_2
obelix[5] > ls -l
-rw-r--r--      2  elsakka  faculty 3456  Sept 10 23:34  readme.txt
-rw-r--r--      1  elsakka  faculty  876  Sept  8 21:02  readme_2.txt
-rw-r--r--      2  elsakka  faculty 3456  Sept 10 23:34  unix_is_easy
lrwxrwxrwx      1  elsakka  faculty   10  Sept 10 23:36  unix_is_easy_2  -> reamme.txt
```


Hard and Symbolic Links

```
obelix[1] > cd
obelix[2] > ln -s  /usr/local/bin  bin.bin
obelix[3] > ls -l
lrwxrwxrwx  1  elsakka  faculty 14 Sept 11 01:34  bin.bin -> /usr/local/bin
……
obelix[4] > cd bin.bin
obelix[5] > pwd
/usr/local/bin
```

Hard and Symbolic Links

- Hard links will have the same authority to a file
  - Removing any one of them, but one, *will not* remove the contents of the file
  - Removing *all* of the hard links *will* remove the contents of the file
- A symbolic link is just an entry pointing to the real name
  - Removing the symbolic link *does not* affect the file
  - Removing the original file will remove the contents of the file and leave the symbolic link pointing to unknown file
- Any user can create symbolic links to directories or files
- Any user can create hard links to files
- Only super users can create hard links to directories
- A hard link must point to a file in the same Unix filesystem
- Before Windows vista, there was no hard link under Windows (Only soft link, a.k.a. symbolic link)
- Even today, you can not create a hard link under windows using GUI (you have to use the Windows text command *mklink /H* to do so)

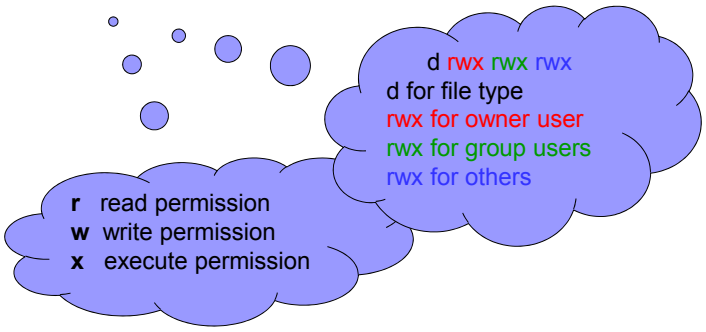
File Permissions

- Unix divides the set of all users in a system into three categories:
  - *owner user*
    - The owner of the file
  - *group users*
    - Users can be in more than one group
    - Each file is assigned to *one* and *only one* group
    - Most of you are in the group 2ndyr
    - Used for easier access control (administration)
    - Normally only the *superuser* can set up groups
  - *other users*
    - Everyone else
- Unix allows you to give *distinct permission privilege* to each category
- The union of *owner user*, *group users*, and *other users* is a universal set
- The intersection between any two groups is an empty set

File Permissions

- Permissions can be viewed with the ls -l command

```
obelix[1] > ls -l
total 1247
-rw------- 1 elsakka  faculty  1117  Sep 14 15:49  bad.cpp
drwx--x--x 2 elsakka  faculty  2048  Oct  11 10:13  prog/
drwxr-xr-x 2 elsakka  faculty   512  Sep 14  23:18  test2/
-rw-r-xr-- 1 elsakka  faculty  2081  Jul   23 15:49  postal_code.txt
```



File Permissions

- Permissions are changed with the chmod command
- There are two syntaxes you can use:

  chmod DDD file [file ...]

  - DDD are 3 ***octal*** digits representing bits of protection
  - rwx rwx rwx can be thought of as 111 111 111 in binary
  - This is the "***numeric***" method
  - It can only perform "***absolute***" permission assignments, e.g.,
    - chmod 644 file

```
rw- r-- r--
110 100 100
 6   4   4
```

File Permissions

chmod [ugoa][+-=][rwx] file [...]

- This is the "***symbolic***" method
- It can perform "***relative***" permission assignments, e.g.,
  - chmod u+rwx file gives the owner user Read, Write, and eXecute
  - chmod g+rx file gives the group users Read and eXecute
  - chmod o-rwx file removes from others Read, Write, and eXecute
  - chmod a+x file gives All eXecute permission
- It can perform "***absolute***" permission assignments, e.g.,
  - chmod g=r file gives group users Read permission and makes sure it has nothing else
- "***relative***" and "***absolute***" permission assignments can be appended with commas (*with no white space before or after the comma*)
  - chmod u=rwx,g-w,o-rwx file

The Umask Command

- umask sets the ***default*** permissions for any file you will create
- The meaning of umask (*un-mask*) value is **opposite** to that of the chmod command
  - tells which permissions will ***not*** be given (i.e., *un-masked*)
    - umask 077 means do ***not*** let anyone but the owner user to do anything with my files by default
- Generally, set umask once in your .cshrc file and never set it again

Directory Permissions

- Directory permissions are ***different*** from the file permissions
  - Listing the contents of the directory requires the read permission to be set
  - Creating files in the directory requires the write permission to be set
  - Accessing files (*if* and *only if* you know its name) in the directory requires the execute permission to be set
- drwx--x--- means it is a directory and it is
  - fully accessible to owner user
  - accessible by name (if you know the name) to group users
  - not accessible to others (including all subdirectories, if any)

Directory Permissions

- The -R option in chmod is useful when working with directories
  - It ***recursively*** changes the mode for each chmod operand that is a directory
  - All sub-directories and files receive those permissions
  - chmod -R a+r dir
    - gives everyone read permission to
      - dir directory and
      - all sub-directories and files under dir
  - chmod -R a+x dir
    - gives the executable access to allow everyone to
      - access dir directory and
      - makes all sub-directories and files under dir executable