Files and Directories

Files and Directories (1)

- What is a file?
  - a container for ordered data
  - persistent (stays around) and accessible by name
- Unix files
  - regular Unix files are pretty simple
    - essentially a sequence of bytes
    - can access these bytes in order
  - Unix files are identified by a name in a directory
    - this name is actually used to resolve the hard disk name/number, the cylinder number, the track number, the sector, the block number
      - you see none of this
    - it allows the file to be accessed

Files and Directories (2)

- Unix files come in other flavors as well, such as
  - Directories
    - a file containing pointers to other files
    - equivalent of a “folder” on a Mac or Windows
  - Links
    - a pointer to another file
    - used like the file it points to
    - similar to “shortcuts” in Windows, but better
  - Devices
    - access a device (like a soundcard, or mouse, or ...) like it is a file

Directories (1)

- Current Working Directory
  - the directory you are looking at right now
  - the shell remembers this for you

- To determine the Current Working Directory, use the command pwd (Print Working Directory)

  Use: obelix[18] > pwd

  Result: print the current working directory

Directories (2)

- Moving about the filesystem
  - Use the "cd" (Change Directory) command to move between directories and change the current directory

    Use: obelix[19] > cd 211

    Result: Makes cs211 the current working directory

- Listing the contents of a directory
  - Use the "ls" (LiSt directory) command to list the contents of a directory

    obelix[20] > ls

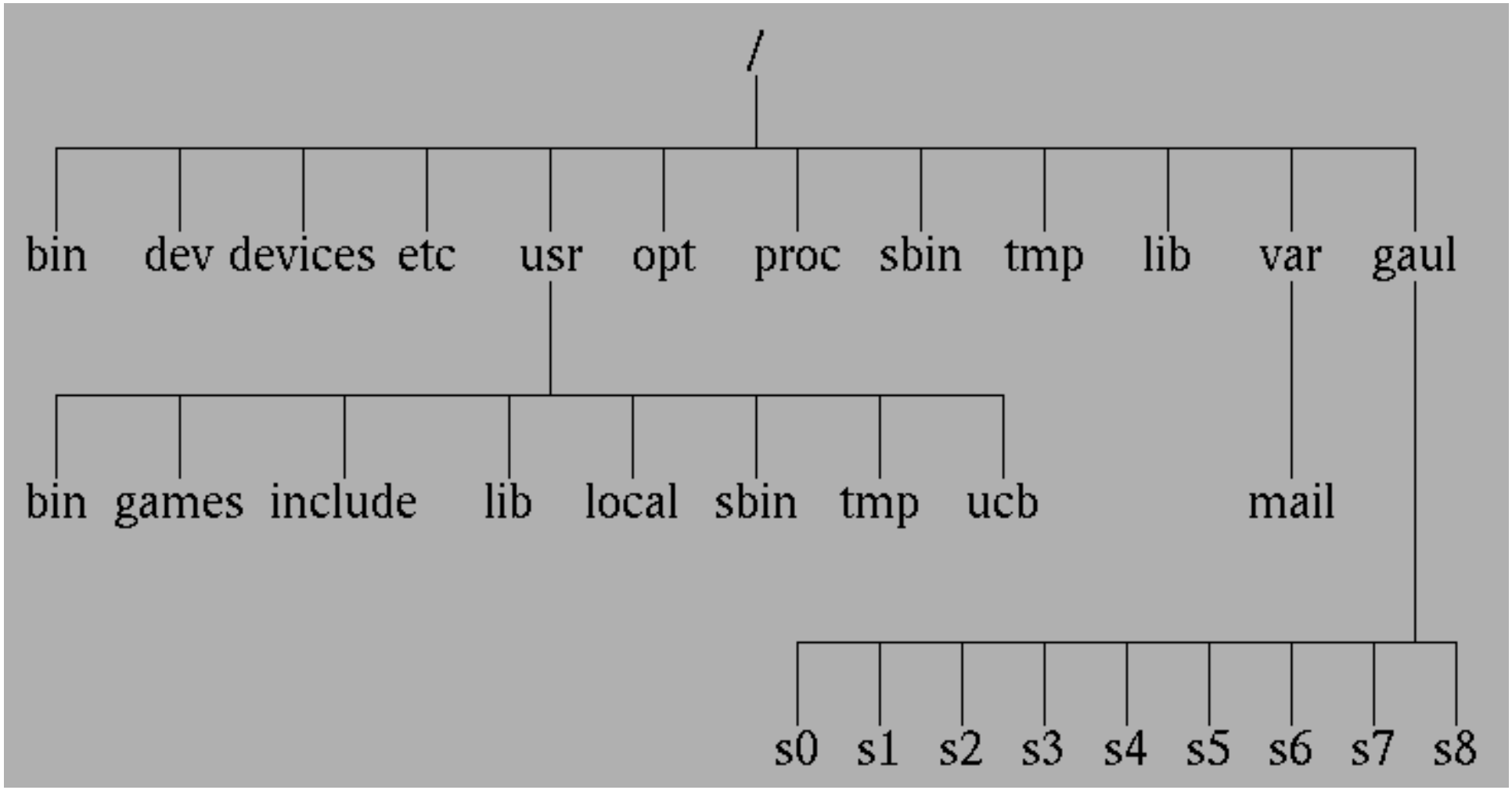
    tmp/ a.out* smit.script cs211@

Directories (3)

- The upside-down tree
  - the Unix filesystem is organized like an upside-down tree
    - at the top of the filesystem is the root
      - write this as a lone slash: **/**
      - this is NOT a backslash (opposite of MS-DOS)!
    - For example, you can change to the root directory:

```
obelix[21] > cd /
obelix[22] > ls
TT_DB/    dev/          home/       mnt/      sbin/    xfn/
bin@      devices/      kernel/     net/      tmp/
cdrom/     etc/         lib@        opt/      usr/
core      export/       local/      platform/  var/
courses@gaul/           lost+found/ proc/      vol/
```

Directories (4)



Directories (5)

- Some standard directories and files in a typical Unix system
  - / the root
  - /bin BINaries (executables)
  - /dev DEVices (peripherals)
  - /devices where the DEVICES really live
  - /etc startup and control files
  - /lib LIBraries (really in /usr)
  - /opt OPTional software packages
  - /proc access to PROCesses
  - /sbin Standalone BINaries
  - /tmp place for TeMPorary files
  - /gaul/ where home directories are mounted
    s0...s9: different places for users

Directories (6)

- /usr USeR stuff
- /usr/bin BINaries again
- /usr/include include files for compilers
- /usr/lib LIBraries of functions etc.
- /usr/local local stuff
- /usr/local/bin local BINaries
- /usr/local/lib local LIBraries
- /usr/openwin X11 stuff
- /usr/sbin sysadmin stuff
- /usr/tmp place for more TeMPorary files
- /usr/ucb UCB binaries
- /var VARiable stuff
- /var/mail the mail spool

Pathnames (1)

- A typical Unix file system spans many disks
  - As a user you don't know or need to know which physical disk things are on
    - in fact, you don't even know which machine they are attached to: disks can be "remote" (eg: your home directory is stored on a disk attached to a server in the machine room)
    - Look at the *df* command to see different disks and space used
  - Inside each directory may be more directories

- The Absolute Path
  - to identify where a file is, string the directories together
    - separating names with slashes:
    - e.g. /gaul/s1/student/1999/csnow
    - this is the absolute path for my home directory
    - lists everything from the root down to the directory you want to specify

Pathnames (2)

- When you first log in, you are in your HOME directory
  - To see what this is:

    obelix[1] > pwd

    /gaul/s1/student/1999/csnow
  - Your home directory is also stored in the environment variable HOME

    obelix[2] > echo My home is $HOME

    My home is /gaul/s1/student/1999/csnow
  - You can “Go Home” by typing

    obelix[3] > cd $HOME

Pathnames (3)

- Some shorthand
  - In some shells (including tcsh, csh, and bash), $HOME can be abbreviated as **~** (tilde)
  - Example: obelix[26] > cd ~/bin
    - change to the bin directory under your home directory (equivalent to $HOME/bin)
    - this is where you usually store your own commands or “executables”
  - To quickly go home:

    obelix[27]% cd

    with no parameters, cd changes to your home directory
  - ~user refers to the home directory of user
    - For me, ~csnow is the same as ~
    - ~doug refers to Doug Vancise’s home directory (/gaul/s1/usr/faculty/doug)

Pathnames (4)

- Relative pathnames
  - You can also specify pathnames relative to the current working directory
    - This is called a relative pathname
  - For example

```
obelix[28] > pwd
 /gaul/s1/student/1999/csnow
obelix[29] > ls
 tmp/      a.out*      smit.script    cs211@
obelix[30] > cd tmp
obelix[31] > pwd
 /gaul/s1/student/1999/csnow/tmp
```

    - Note: You don't need to know absolute pathnames
- For most commands which require a file name, you can specify a pathname (relative or absolute)

Pathnames (5)

- Every directory contains two “special” directories: . and ..
  - . : another name for the current directory
    - e.g. cp cs211/foo .
  - .. : another name for the immediate parent directory of the current directory
    - use this to cd to your parent:

```
obelix[32] > pwd
 /gaul/s1/student/1999/csnow
obelix[33] > cd ..
obelix[34] > pwd
 /gaul/s1/student/1999
obelix[35] > cd ../..
obelix[36] > pwd
 /gaul/s1
```

Pathnames (6)

- You can locate a file or directory by this way:
  - look at the first character of the pathname
    - / start from the root
    - . start from the current directory
    - .. start from the parent directory
    - ~ start from a home directory
    - else start from the current directory
  - going down to the subdirectories in the pathname, until you complete the whole pathname.
  - if you start in ~csnow, the following are equivalent:
    - /gaul/s1/student/1999/csnow/cs211/readme.txt
    - ~/cs211/readme.txt
    - cs211/readme.txt

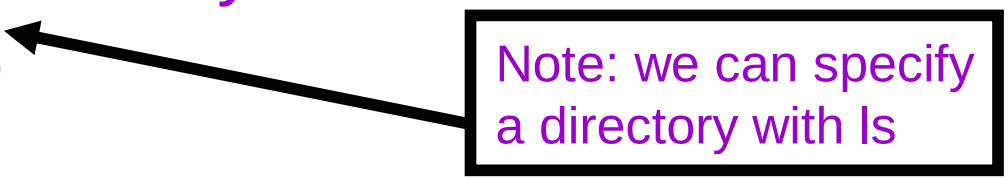
Working with Directories (1)

- Create a directory with the mkdir command

  mkdir newdirname

- newdirname can be given with pathname

```
obelix[37] > pwd
/gaul/s1/student/1999/csnow/cs211
obelix[38] > ls
readme.txt
obelix[39] > mkdir mydir1
obelix[40] > ls
readme.txt   mydir1/
obelix[41] > mkdir mydir1/mydir2
obelix[42] > ls mydir1
mydir2/
obelix[43] > cd mydir1/mydir2
```



Working with Directories (2)

- Remove a directory with the rmdir command

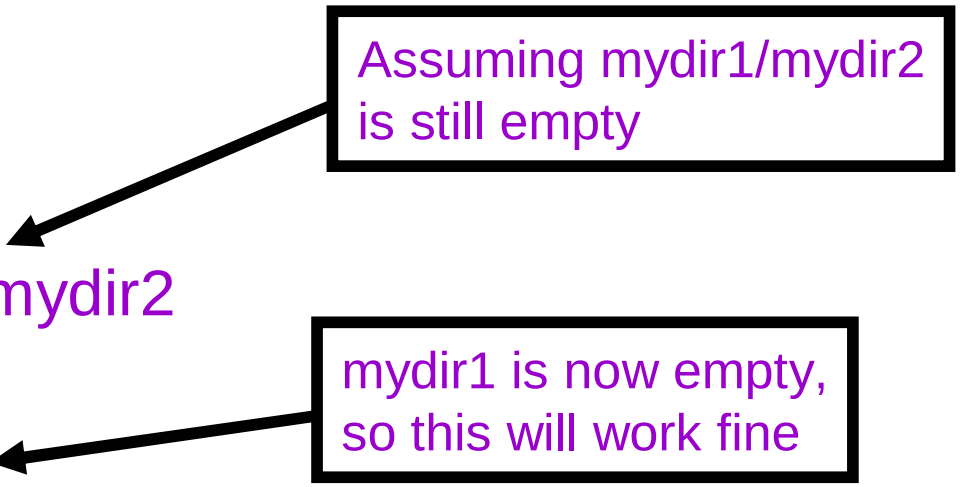
  rmdir dirname

  – dirname is the directory to remove and can be specified using a pathname

  – if the directory exists and is **empty** it will be removed

- Examples:

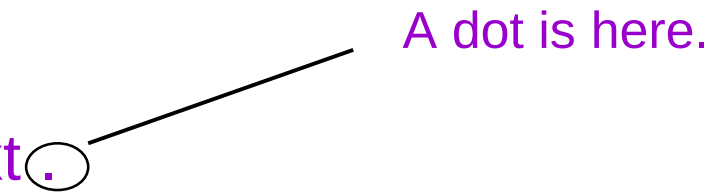
```
obelix[44] > cd ~/cs211; ls
readme.txt  mydir1/
obelix[45] > ls mydir1
mydir2/
obelix[46] > rmdir mydir1/mydir2
obelix[47] > ls mydir1
obelix[48] > rmdir mydir1
```



Working with Directories (3)

- Move a file from one directory to another

```
obelix[1] >  pwd
/gaul/s1/student/1999/csnow/cs211
obelix[2] >  ls
readme.txt  mydir1/
obelix[3] >  ls mydir1
hello.txt
obelix[4] >  mv mydir1/hello.txt .
obelix[5] >  ls mydir1
obelix[6] >  ls
readme.txt hello.txt  mydir1/
```



- You can also move a directory the same way - it is just a special file, after all.

Working with Directories (4)

- Copy a file from one directory to another

```
obelix[1] > ls
readme.txt  mydir1/
obelix[2] > cp readme.txt mydir1
obelix[3] > ls mydir1
readme.txt
```

- Copying a directory

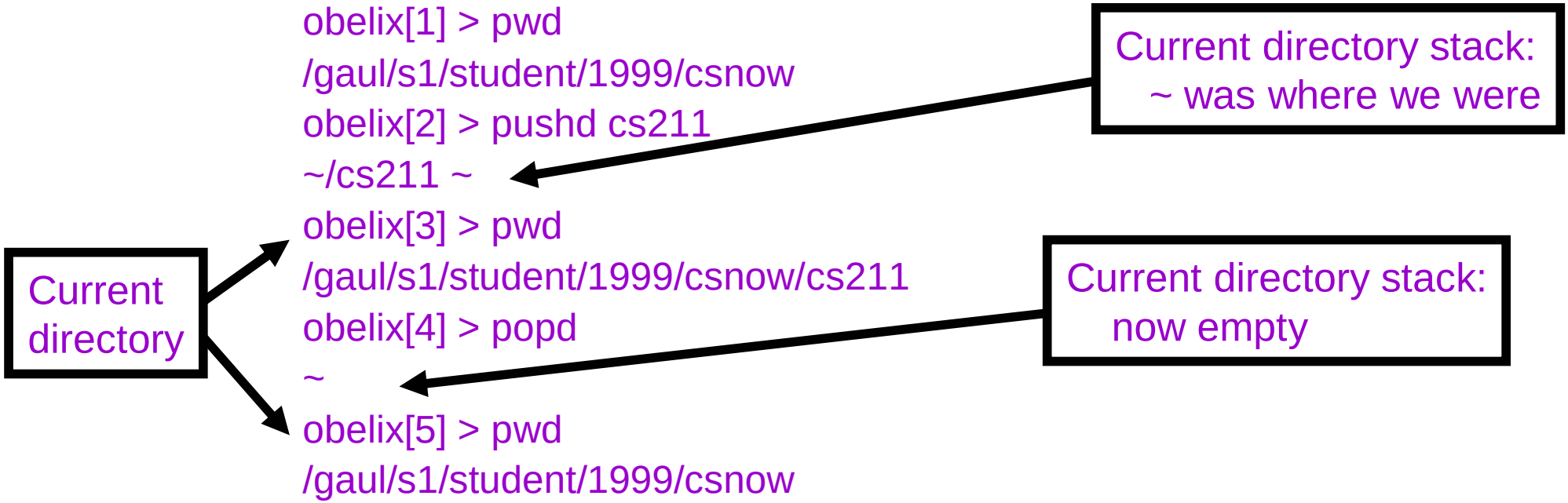
```
obelix[4] > cp mydir1 mydir2
cp: mydir1: is a directory
obelix[5] > cp -r mydir1 mydir2
obelix[6] > ls mydir2
readme.txt
```

Cannot use just cp to copy a directory

Must do a recursive copy (cp -r) to copy a directory

Working with Directories (5)

- Some shells (csh and tcsh) provide pushd and popd directory commands
- pushd changes directories, but remembers the previous one by pushing it on to a stack
- popd changes directories back to the last directory placed on the stack by pushd



Working with Directories (6)

- What if you need to locate a file, or set of files, in a large directory structure?
  - Using cd and ls would be very tedious!
- The command find is used to search through directories to locate files.
  - Wildcards can be used, if the exact file name is unknown, or to find multiple files at once.
  - Can also find files based on size, owner, creation time, type, permissions, and so on.
  - Can also automatically execute commands on each file found.
- Do a "man find" for details and examples!

More Files and Directories (1)

- What files do I already have?
  - Startup files for csh and tcsh (.login, .cshrc)
  - Contain commands run after you type your password, but before you get a prompt
  - Assume you've not used your account before

    obelix[1] > ls

    obelix[2] >
  - Why can't I see any files?
    - Files beginning with a 'dot' are usually control files in Unix and not generally displayed
  - Use the –a option to see all files

    obelix[3] > ls -a

    ./ ../ .cshrc .login

    obelix[4] >

More Files and Directories (2)

- OK, let us study some new commands, and variations of some familiar ones

```
obelix[51] > ls -a
./       ../      .cshrc    .login
obelix[52] > cp .cshrc my_new_file
obelix[53] > ls -a
./           ../          .cshrc       .login       my_new_file
obelix[54] > cp -i .login my_new_file
cp: overwrite my_new_file (yes/no)? y
obelix[55] > head –7 my_new_file
#
#  WGUI is twm or mwm
#

if (!($?HOSTTYPE)) then
  set HOSTTYPE = `uname -m`
endif
```

list all files including those beginning a with .

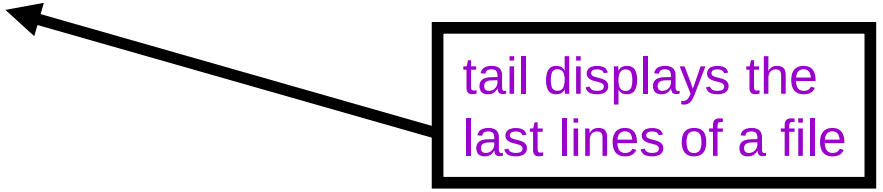
The –i option says to ask when this overwrites existing files.

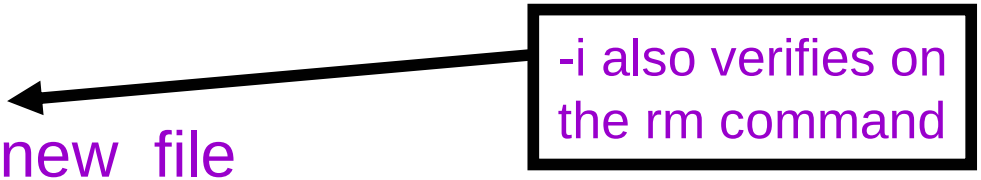
head displays the top lines of a file

More Files and Directories (3)

```
obelix[56] > tail -8 my_new_file
breaksw
#
 default:
   echo "** .login: Unknown Host Type **"
 breaksw

endsw
obelix[57] > rm -i my_new_file
rm: remove my_new_file (yes/no)? y
obelix[58] > ls –a
./          ../          .cshrc       .login
```





Unix Filenames (1)

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

Unix Filenames (2)

- No enforced extensions
  - The following are all legal Unix file names
    - a
    - a.
    - .a
    - …
    - a.b.c
- Remember files beginning with dot are hidden
  - ls cannot see them, use ls -a
- . and .. are reserved for current and parent directories

Unix Filenames (3)

- Even though Unix doesn't enforce extensions,
  - “.” and an extension are still used for clarity
    - .jpg for JPEG images
    - .tex for LaTeX files
    - .sh for shell scripts
    - .txt for text files
    - .mp3 for MP3’s
  - some applications may enforce their own extensions
    - Compilers look for these extensions by default
      - .c means a C program file
      - .C or .cpp or .cc for C++ program files
      - .h for C or C++ header files
      - .o means an object file

Unix Filenames (4)

- Executable files usually have no extensions
  - cannot execute file a.exe by just typing a
  - telling executable files from data files can be difficult
- "file" command

  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
  - 1024 characters typically
  - these are pretty long (much better than MS-DOS days and the 8.3 filenames)

Fixing Filename Mistakes

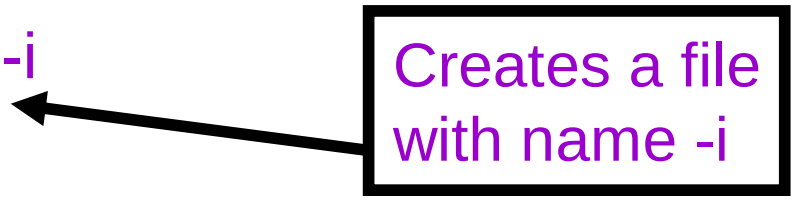
- It is very easy to get the wrong stuff into filenames
  - Say you accidentally typed

    obelix[3] > cp myfile -i

  - What if you type

    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 (1)

- 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 one
  - a sequence may also match a large number of sequences
- Often use wildcard characters to match filenames
  - filename substitution – generally known as “**globbing**”
- Wildcard characters

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

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

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

Filename Wildcarding (2)

- 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: csnow*c does not match csnow/codec
- Wildcards are expanded by the shell, and not by the program
  - Programmers of commands do not worry about searching the directory tree for matching file names
  - The program just sees the list of files matched

Filename Wildcarding (3)

- Matching the dot
  - A dot (**.**) at
    - the beginning of a filename, or
    - immediately following a /

    must be matched explicitly.
  - Similar to the character ***/***
  - Example:

    obelix[8] > cat **.**c*

cat all files whose names begin with .c

- As mentioned earlier, [....] matches any one of the characters enclosed
  - Within "[...]", a pair of characters separated by "-" matches any character lexically between the two
    - Example:

      obelix[9] > ls [a-z]*

lists all files beginning with a character between ASCII 'a' and ASCII 'z'

Filename Wildcarding (4)

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

    obelix[10] > ls /bin/*[-_]*
  - What about this?

    obelix[11] > ls *
  - What about this?

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

    Answer: this one is complicated…

Unix Quoting (1)

- Double Quotes: "...."
  - Putting text in double quotes "..." stops interpretation of some shell special characters (whitespace mostly)
  - Examples:

```
obelix[12] > echo  Here    are       some       words
Here are some words
obelix[13] > echo "Here    are       some       words"
Here    are       some       words
obelix[14] > mkdir "A directory name with spaces! "
obelix[15] > ls A*
A directory name with spaces!/
```

Unix Quoting (2)

- Single Quotes '...'
  - Stops interpretation of even more specials
    - Stop variable expansion ($HOME, etc.)
    - Backquotes `...` (execute a command and return result ...we'll get to this later)
    - Note difference: single quote ('), backquote (`)
    - Examples:

```
obelix[16] > echo "Welcome $HOME"
Welcome /gaul/s1/student/1999/csnow
obelix[17] > echo ‘Welcome $HOME’
Welcome $HOME
```

Unix Quoting (3)

- Backslash ****
  - ‘quotes’ the next character
  - Lets one escape all of the shell special characters

```
obelix[18] > mkdir Dir\ name\ with\ spaces\*\*
obelix[19] > ls Dir\ *
Dir name with spaces**/
```

  - Use backslash to escape a newline character

```
obelix[20]% echo "This is a long line and\
we want to continue on the next“
This is a long line and we want to continue on the next
```

  - Use backslash to escape other shell special chars
    - Like quote characters

```
obelix[21] > echo \"Bartlett\'s Familiar Quotations\"
"Bartlett's Familiar Quotations"
```

Unix Quoting (4)

- Control-V
  - Quotes the next character, even if it is a control character
  - Lets one get weird stuff into the command line
  - Very similar to backslash but generally for ASCII characters which do not show up on the screen
  - Example: the backspace character

```
obelix[22] > echo "abc^H^H^Hcde"
cde
```

Control-h is backspace on most terminals

typing Control-v Control-h enters a "quoted" Control-h to the shell
• written ^H

  - Precisely how it works is dependant on the shell you use, and the type of terminal you are using

Hard and Symbolic Links (1)

- When a file is created, there is one link to it.
- Additional links can be added to a file using the command ln. These are called hard links.
- Each hard link acts like a pointer to the file and are indistinguishable from the original.

```
obelix[1] > ls
readme.txt
obelix[2] > ln readme.txt unix_is_easy
obelix[3] > ls
readme.txt   unix_is_easy
```

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

Hard and Symbolic Links (2)

- A symbolic link is an indirect pointer to another file or directory.
- It is a directory entry containing the pathname of the pointed to file.

```
obelix[1] > cd
obelix[2] > ln -s  /usr/local/bin  bin
obelix[3] > ls -l
lrwxrwxrwx  bin -> /usr/local/bin
……
obelix[4] > cd bin
obelix[5] > pwd
/usr/local/bin
```

Hard and Symbolic Links (3)

- Two hard links have the same authority to a file
  - Removing any of them will NOT remove the contents of the file
  - Removing all of the hard links will remove the contents of the file from the hard disk.
- A symbolic link is just an entry to the real name
  - Removing the symbolic link does not affect the file
  - Removing the original name will remove the contents of the file
- Only super users can create hard links for directories
- Hard links must point to files in the same Unix filesystem

FTP (1)

- FTP File Transfer Protocol

```
obelix[1] > ftp gaul.csd.uwo.ca
Connected to gaul.csd.uwo.ca
220 gaul.csd.uwo.ca FTP server ready.
Name : csnow
331 password required for csnow
Password:
230 user csnow logged in.
ftp> get remotefile localfile
……..
ftp> quit
221 Goodbye
obelix[2] >
```

FTP (2)

- Basic FTP commands
  - ls list the remote directory (dir is more verbose)
  - cd change the remote directory
  - get remotefile [localfile] download remotefile
  - put localfile [remotefile] upload localfile
  - bye quit
  - ? list all the available commands
  - ? command print the information about a command
  - mget file_name_with_wildcards get multiple files
  - mput file_name_with_wildcards put multiple files
  - prompt toggles prompting with mget and mput
  - bin transfer binary files (8 bits per char)