Files and Directories

Files and Directories (1)

◆ What is a file?
- ▪ a container for data
- ▪ persistent (stays around) and accessible by name

◆ Unix files
- ▪ Unix file types: regular, directory, device, link and etc.
- ▪ 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)

◆ **Regular file**

- essentially a sequence of bytes

- can access these bytes in order

- Text file: contains only printable characters and Line Feed ("LF") or *newline* character

- Binary file: contains characters of entire ASCII range from 0 to 255

◆ **Directory file**

- a file containing name and pointer (inode number) pairs for files and subdirectories in the directory

- equivalent of a "folder" in Windows

Files and Directories (3)

◆ **Device file**

- access a device (like a soundcard, mouse, terminal, or ...) like it is a file

◆ **Links**

- hard link: create another name for a file
- soft link: a pointer to another file
  - used like the file it points to
  - similar to "shortcuts" in Windows

◆ **FIFO** and etc.

Files and Directories (4)

◆ To get file type information

  – Use –F option for ls

    compute[1] > ls –F

    tmp/   a.out*   smit.script   cs211@

        | Directories | | Executable | | Link |

  – Use ls --color=auto

    different file type with different color

  – Use ls –l

    first letter of the long listing format indicates file type
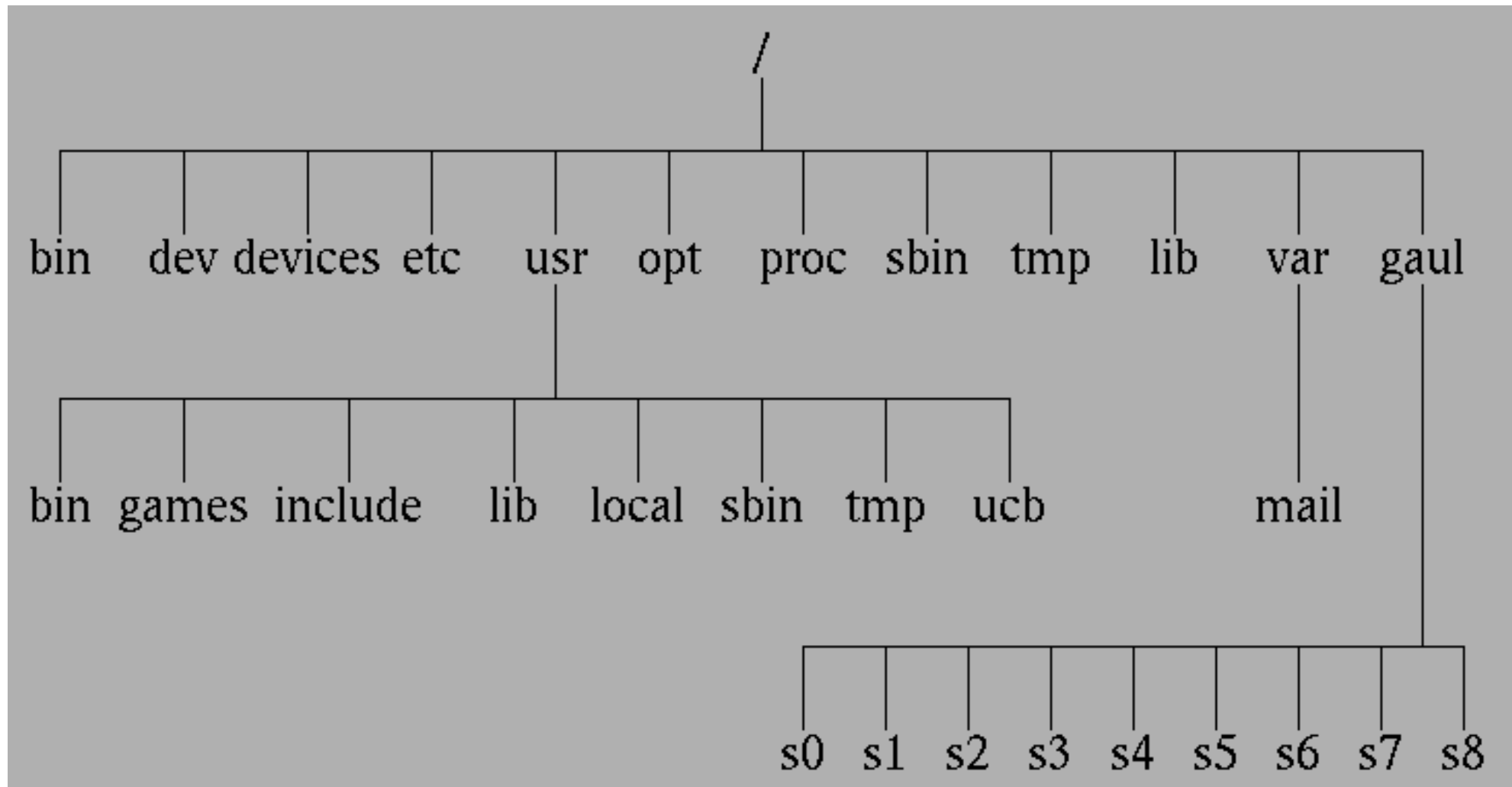
Unix File System (1)

◆ The upside-down tree

  – the Unix file system is organized like an upside-down tree

     ❖ at the top of the file system 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:

```
compute[1] > cd /
compute[2] > ls
TT_DB/      dev/        home/         mnt/        sbin/       xfn/
bin@        devices/   kernel/        net/        tmp/
boot/       etc/        lib@          opt/        usr/
core        export/     lib64@        platform/   var/
courses@   gaull/      lost+found/   proc/       vol/
```

Unix File System (2)

Unix File System (3)

◆ Some standard directories and files in a typical Unix system

– /   the root
– /bin   BINaries executables, i.e. commands
– /dev   DEVices (peripherals)
– /sys/devices   where the DEVICES really live
– /etc   startup and control files
– /lib   LIBraries
– /opt   OPTional software packages
– /proc   access to PROCesses
– /sbin   Standalone BINaries
– /tmp   place for TeMPorary files
– /usr   USeR stuff

Unix File System (4)

- /usr/bin   BINaries (commands) 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/sbin   sys admin stuff
- /usr/tmp   place for more TeMPorary files
- /usr/ucb   UCB binaries
- /var   VARiable stuff
- /var/mail   the mail spool

Unix File System (5)

◆ 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" (e.g.: your home directory is stored on a disk attached to a server in the machine room.

- Part of the file system may come from another system with network file system

  ❖ Look at the *df* command to see different disks or different filesystems and space used.

- You just need to know the tree structure of the file system

Pathnames (1)

◆ **In Unix file system, inside each directory there may be more directories**

◆ **The Absolute Path**

– to identify where a file is, concatenate the directories together

❖ separating names with slashes:

❖ e.g. /home/kzhang

❖ 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:

compute[1] > pwd

/home/kzhang

– Your home directory is also stored in the environment variable HOME

compute[2] > echo My home is $HOME

My home is /home/kzhang

– You can "Go Home" by typing

compute[3] > cd $HOME

Pathnames (3)

◆ Some shorthand

– In some shells (including bash, tcsh, and csh), $HOME can be abbreviated as **~** (tilde)

– Example: compute[4] > 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:

  compute[5] > cd ~   or just   cd

  with no parameters, cd changes to your home directory

– ~user refers to the home directory of user

  ❖ For me, ~kzhang is the same as ~

  ❖ ~doug refers to Doug Vancise's home directory (/home/doug)

Pathnames (4)

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

      compute[6] > pwd
       /home/kzhang
      compute[7] > ls
       tmp/   a.out*   smit.script   cs211@
      compute[8] > cd tmp
      compute[9] > pwd
       /home/kzhang/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 **..**   Use ls –a to see them

  - **.** : another name for the current directory

    - – e.g.   cp cs2211/foo .

  - **..** : another name for the immediate parent directory of the current directory

    - – use this to cd to your parent:

      compute[10] > pwd

      /home/kzhang/cs2211

      compute[11] > cd ..

      compute[12] > pwd

      /home/kzhang
      compute[13] > cd ../..
      compute[14] > pwd
      /

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 a home directory
    ❖ **.**   start from the current directory
    ❖ **..**   start from the parent 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 ~kzhang, the following are equivalent:
    ❖ /home/kzhang/cs2211/readme.txt
    ❖ ~/cs2211/readme.txt
    ❖ cs2211/readme.txt

Working with Directories (1)

◆ **Home Directory**

   – When a user log in, the login directory is the user's home directory

◆ **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: compute[1] > pwd

   – result: print the current working directory

Working with Directories (2)

◆ Create a directory with the mkdir command

mkdir newdirname

◆ newdirname can be given with pathname

compute[2] > pwd

/home/kzhang/cs2211

compute[3] > ls

readme.txt

compute[4] > mkdir mydir1; ls

readme.txt   mydir1/
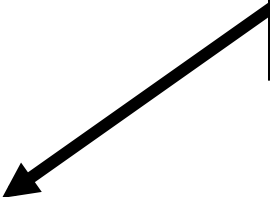
compute[5] > mkdir mydir1/mydir2

compute[6] > ls mydir1

mydir2/

compute[7] > cd mydir1/mydir2

Note: two commands
in the same line

Note: we can specify
a directory with ls

Working with Directories (3)

◆ 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 **<u>empty</u>** it will be removed

◆ Examples:

> compute[8] > cd ~/cs2211; ls
> readme.txt mydir1/
> compute[9] > ls mydir1
> mydir2/
>
> compute[10] > rmdir mydir1/mydir2
>
> compute[11] > ls mydir1
>
> compute[12] > rmdir mydir1

Assuming mydir1/mydir2 is still empty

mydir1 is now empty, so this will work fine

Working with Directories (4)

◆ Move a file from one directory to another

compute[13] > pwd; ls; ls mydir1

/home/kzhang/cs2211

mydir1/ readme.txt

hello.txt

compute[14] > mv mydir1/hello.txt . ← A dot is here.

compute[15] > ls mydir1

compute[16] > ls

hello.txt mydir1/ readme.txt

◆ Rename with mv

compute[17] > mv hello.txt world.txt

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

Working with Directories (5)

◆ Copy a file from one directory to another

compute[18] > ls

readme.txt mydir1/

compute[19] > cp readme.txt mydir1

compute[20] > ls mydir1

readme.txt

◆ Copying a directory

compute[21] > cp mydir1 mydir2

cp: mydir1: is a directory

compute[22] > cp –r mydir1 mydir2

compute[23] > ls mydir2

readme.txt

◆ What if mydir2 already exist?

Cannot use just cp to copy a directory

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

Working with Directories (6)

◆ Some shells (bash, 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

compute[24] > pwd

/home/kzhang

compute[25] > pushd cs2211

~/cs2211 ~

compute[26] > pwd

/home/kzhang/cs2211

compute[27] > popd

~

compute[28] > pwd

/home/kzhang

Current directory stack:
~ was where we were

Current directory stack:
now empty

Current directory

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.

compute[1] > ls

readme.txt

compute[2] > ln readme.txt unix_is_easy

compute[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 file.

compute[1] > cd

compute[2] > ln –s /usr/local/bin bin

compute[3] > ls –l

lrwxrwxrwx bin –> /usr/local/bin

……

compute[4] > cd –P bin

compute[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

Finding Files (1)

- 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!

Finding Files (2)

- Use the find command
  - Recursively searches, starting from a given directory, for files and/or directories matching a given expression

- Usage: find PATH EXPRESSION.
  - e.g. Find all files and directories named README starting from the current directory:

    find . –name "README"

  - e.g. Find all files and directories with the extension .h starting from /usr/include:

    find /usr/include –name "*.h"   * is called a wildcard

Finding Files (3)

- Usage: find PATH EXPRESSION.
  - e.g. Find all regular files (but not directories) named backup starting from the current directory:

    find . –type f –name "backup"

  - e.g. Find all C programs of my account:

    find ~ –type f –name "*.c"

  - e.g. Find all directories (but not other files) named backup starting from the current directory:

    find . –type d –name "backup"

More Files and Directories (1)

◆ What files do I already have in my home directory?

– May have startup files for bash (.bash_profile, .bashrc)

– Contain commands run after you type your password, but before you get a prompt

– Assume you've not used your account before

compute[1] > ls

compute[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

compute[2] > ls –a

./   ../   .bash_history .bash_profile .bashrc   …   .viminfo

– May have startup files for csh and tcsh (.login, .cshrc)

More Files and Directories (2)

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

```
compute[3] > ls  –a
./  ../  .bash_history .bash_profile .bashrc  …
compute[4] > cp .bashrc my_new_file
compute[5] > ls  –a
./   ../   .bash_history .bash_profile .bashrc   …   my_new_file
compute[6] > cp  –i .bash_profile my_new_file
cp: overwrite my_new_file (yes/no)? y
compute[7] > tail  –7  my_new_file

export PS1="\h:\w[\!] > "

# get aliases and functions
if [ –f  ~/.bashrc ]; then
  .  ~/.bashrc
fi
```

list all files including those beginning with a .

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

tail displays the bottom lines of a file

More Files and Directories (3)

compute[8] > head –6 my_new_file

set PATH and/or TERM

export PATH=$PATH:$HOME/bin:.

> head displays the top lines of a file

set umask

umask 077

> –i also verifies on the rm command

compute[9] > rm –i my_new_file

rm: remove my_new_file (yes/no)? y

You may want to put
 alias rm='rm –i'
in your .bashrc file

More Files and Directories (4)

- Pager
  - A command that displays one screen of text at a time
  - Waits for user to press a key and then displays the next screen

- more
  - Can only move forward.
  - Press the **Spacebar** to move to next screen.
  - Press **q** at any time to quit.
  - e.g. Display **README.txt**: **more README.txt**

More Files and Directories (5)

- **less**
  - Can move forward and backward
  - Use arrow keys to move up and down one line at a time
  - Press the **Spacebar** to move to next screen
  - Press **Ctrl+b** to move back one screen
  - Press **q** at any time to quit
  - e.g. Display **README.txt**: **less README.txt**

Unix File Names (1)

- ◆ Almost any character is valid in a file name
  - ▪ all the punctuation and digits
  - ▪ the exception is the / (slash) character and null character
  - ▪ usually letters, digits, and . – _
  - ▪ the following are not encouraged
    - ❖ ? * [ ] " " ' ( ) & : ; !
  - ▪ the following are not encouraged as the first character
    - ❖ – ~
  - ▪ control characters are also allowed, but are not encouraged

Unix File Names (2)

◆ Upper and lower case letters are different

   – A.txt and a.txt are different files

◆ 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

Unix File Names (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

      ❖ .mp4 for MP4'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 File Names (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: compute[1] > file .bashrc
                 .bashrc: ASCII text

◆ Filenames and pathnames have limits on lengths

– 255 for file name and 4096 for path 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

compute[2] > cp -- myfile –i

Creates a file
with name –i

– What if you type

compute[3] > 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

compute[4] > 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 file names

– Filename expansion – generally known as "globbing"

◆ Filename expansion 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

compute[1] > mv a*.[ch] cfiles/

❖ mv all files beginning with a and ending with .c or .h into the directory cfiles

compute[2] > 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

– Filename expansion will only do expansion in one directory

◆ Wildcards are expanded by the shell, and not by commands

– Programmers of commands do not worry about searching the directory tree for matching file names

– The program just sees the list of files matched

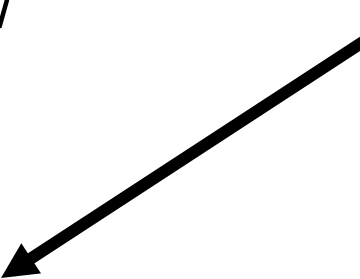# Filename Wildcarding (3)

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

    must be matched <u>explicitly</u>.
  - – Similar to the character **/**
  - – Example:

    compute[3] > cat *profile

    Cat will not find
    .bash_profile

- ◆ 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:

    compute[4] > ls [a-z]*

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

    (may or may not work for default bash)

Filename Wildcarding (4)

◆ More advanced examples:

   – What does the following do?

      compute[5] > ls /bin/*[–_]*

   – What about this?

      compute[6] > ls *

   – What about this?

      compute[7] > 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:

  compute[1] > echo Here   are   some   words

  Here are some words

  compute[2] > echo "Here   are   some   words"

  Here   are   some   words

  compute[3] > mkdir "A directory name with spaces! "

  compute[4] > echo "Welcome $HOME"

  Welcome /home/kzhang

Unix Quoting (2)

◆ Single Quotes **'...'**

  – Stops interpretation of even more specials

    ❖ stop variable expansion ($HOME, etc.)

      compute[5] > echo 'Welcome $HOME'

      Welcome $HOME

◆ Back Quotes **`...`**

  – execute a command and return result

      compute[6] > echo `ls  –d c*`

      cs2211

◆ Note difference: single quote (**'**), back quote (**`**)

Unix Quoting (3)

◆ Backslash \

– 'quotes' the next character

– Lets one escape all of the shell special characters

compute[7] > mkdir Dir\ name\ with\ spaces**

compute[8] > ls Dir\ *

Dir name with spaces**/

– Use backslash to escape other shell special characters

❖ Like quote characters

compute[9] > echo \"Bartlett\'s Familiar Quotations\"

"Bartlett's Familiar Quotations"

– Use backslash to escape a newline character

compute[10] > 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

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

    compute[11] > 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 dependent on the shell you use, and the type of terminal you are using

SFTP (1)

◆ SFTP Secure File Transfer Protocol

compute[1] > sftp kzhang@compute.csd.uwo.ca

Password:

Connected to compute.csd.uwo.ca.

sftp> get remotefile localfile

……….

sftp> quit

What if I do the following?

compute[2] > ssh kzhang@compute.gaul.csd.uwo.ca

……….

compute[2] > sftp kzhang@compute.gaul.csd.uwo.ca

SFTP (2)

◆ Basic SFTP commands
  – ls   list the remote directory (lls for local directory)
  – pwd   show path of the remote directory (lpwd)
  – cd   change the remote directory (lcd)
  – get remotefile [localfile]   download remotefile
  – put localfile [remotefile]   upload localfile
  – get file_name_with_wildcards   get multiple files
  – put file_name_with_wildcards   put multiple files
  – bye   quit
  – quit   quit
  – ?   list all the available commands