Twwong9 Talk Preferences Watchlist Contributions Log out

Page Discussion Read Edit View history More Search CDOT Wiki Go

CDOT
SICT AR Meeting Area
People

get involved with CDOT
as a Student
as an Open Source Community Member
as a Company

courses
BTC640
BTH740
BTP300
DPI908
DPS901
DPS905
DPS909
DPS911
DPS914
DPS915
DPS924
DPS931
EAC234
ECL500
GAM531
GAM666
GAM670
GPU610
LUX Program
MAP524
OOP344
OPS235
OPS245
OPS335
OPS345
OPS435
OPS445
OPS535
OPS635
OSD600
OSD700
OSL640
OSL740
OSL840

ULI101 Week 3

Contents [hide]

Pathnames [edit]

- A pathname is a list of names that will lead to a file.
- Essentially they are directories, but a file name itself is a path as well
- The concept of a pathname relates to every operating system including Unix, Linux, MS-DOS, MS-Windows, Apple-Macintosh, etc.

```
# Directory pathname:
/home/username/ics124/assignments/

# File pathname:
/home/username/ops224/assignments/assn1.txt
```

Absolute vs Relative Pathnames [edit]

Absolute Pathname [edit]

- A pathname that begins from root.

- The pathname begins with a forward slash, for example: `/home/someuser/unx122`

Relative Pathname [edit]

- A pathname that is "relative" to the location of the current or "working" directory.
- For example, if we are in your home directory, issuing the command `mkdir uli101` will create the uli101 directory in your home directory.
- Rules:

  1. A relative pathname does NOT begin with a slash.
  2. A relative pathname can use the following symbols can be used at the beginning:
     - `..` parent directory (up one directory level)
     - `.` current directory
  3. Not all relative pathnames begin with `..`
  4. When using relative pathname, make sure you know your present working directory.

```
# Change pwd to ipc144 (.. means parent directory of pwd)
cd ../ipc144

# copy file sample.c (location is relative to parent of
pwd)
# from joe.doe's home directory to your pwd (. means pwd)
cp ../joe.doe/sample.c .
```

Relative-to-Home Pathnames [edit]

You can specify a pathname as relative-to-home by using a tilde and slash at the start, e.g. `~/uli101/notes.html`. The tilde ~ is replaced by your home directory (typically *home/your.account*) to make the pathname absolute. You can immediately place a username after the tilde to represent another user's home directory, for example:

- `~jane.somebody` expands to `/home/jane.somebody` but `~` expands to `/home/your_home_dir`
- similarly `~uli101` expands to `/home/uli101` but `~/uli101` expands to `/home/your_home_dir/uli101`

Which Type of Pathname to Use? [edit]

So far, we have been given many different types of pathnames that we can use for regular files and directories:

- Absolute pathname (starts with `/`)
- Relative pathname (doesn't start with `/`)
- Relative-to-home pathname (starts with `~`)

You can decide which pathname type to use to make it more convenient (eg relative - usually less typing or absolute if you don't know what directory you are currently located in).

Making Directories [edit]

Building directories is similar in approach to building a house:

- Begins from a foundation (eg home directory).
- Need to build in proper order (add on addition to house in right location). Use a logical scheme.
- When building directories from different locations, must provide proper absolute or relative pathname.

Planning Directories [edit]

Good directory organization requires planning:

- Group information together logically.
- Plan for the future: use dated directories where appropriate (`~/christmas/2001`, `/christmas/2002`)
- Too few directories = excessive number of files in each; too many directories = long pathnames.

Where to build directories? [edit]

- Want to build a directory called tmp that branches off of your home directory?
- Verify which directory you are located (either look at directory from command prompt or issue the command pwd)
- Type mkdir tmp at the Unix prompt, followed by ENTER
- Optionally you can verify that directory has been created using ls or ls -ld commands)

Creating Parent Directories [edit]

By default, a directory cannot be created in a nonexistent location - it needs a parent directory To create directory paths with parent directories that do not exist (using a single command) use the `-p` option for the mkdir command

```
# This would create the parent directory mur and then the child
directory dir1.
# The -p means "create all the directories in the Path".
$ mkdir -p mur/dir1
```

Removing Directories [edit]

Removing directories is reverse order of building directories

- Issue command rmdir directory
- rmdir cannot remove directories containing files or other subdirectories.
- rmdir cannot remove directories that are anyone's current directory.
- Need to step back to at least parent directory to remove an empty directory.

Removing Sub-trees [edit]

- To remove a sub-tree (a directory and all of its contents including sub-directories) use `rm -r` directory (or `rm -R` directory).

The can use the `rm -rf` command (`-f = force`) to complete delete files and directories recursiverly, even if they are protected from delete

* Remove files only if you are absolutely sure what you are doing.

- rm -r can erase large numbers of files very quickly. Use with extreme care!
- Backing up your data is a very good idea.

Filename Expansion [edit]

- Many of the commands discussed so far make reference to a specific filename - e.g. and regular file to store data, or a directory.
- Sometimes the user may not know the exact name of a file, or the user wants to use a command to apply to a number of files that have a similar name

For example: `work.txt, work2.txt, work3.txt`

- Special characters can be used to expand a general filename and use them if they match. You may have heard about "Wildcard Characters" - this is a similar concept.
- Symbols:

* (star/asterisk)	Represents zero or more of any characters.
? (question mark)	Represents any single character.
[ ] (character class)	Represents a single character, any of the list inside of the brackets. Placing a ! Symbol after first square bracket means "opposite"). Ranges such as [a-z] or [0-3] are supported.

- To demonstrate filename expansion, let's assume the following regular files are contained in our current directory:

```
$ touch work1.txt work2.txt work3.txt work4.c worka.txt
working.txt
$ ls
work1.txt work2.txt work3.txt work4.c worka.txt working.txt
```

- Note the results from using filename expansion:

```
$ ls work*
work1.txt work2.txt work3.txt work4.c worka.txt working.txt

$ ls work?.txt
work1.txt work2.txt work3.txt worka.txt

$ ls work[1-3].txt
work1.txt work2.txt work3.txt

$ ls work[!1-3].txt worka.txt
```

UNIX shell [edit]

- Command interpreter for UNIX
- Acts as a mediator between user and UNIX kernel
- Processes and/or executes user commands
- More than one command can be executed on one command line when separated by a semi-colon
- You will be learning approx. 30 Unix commands in this course. This is a small, compared to the the 1000+ Unix commands out there
- The term command and utility mean the same in Unix UNIX shell
- There are several kinds of shells available for UNIX
- Most popular shells are:

C shell (this is not the C programming language)

Korn shell - used with Unix

Linux machines most often use the BASH shell (Bourne-Again Shell)

- Each user on one machine can run a different shell
- UNIX scripting = UNIX shell programming

Why command line? [edit]

- Why don’t we just use the GUI (KDE, Gnome or some other window manager)? - GUI may not always be available
- What if something is broken?
- What if you are connecting through a terminal remotely? - GUI is for regular users
- Many administrative tools are hard to find in the menus - Command line is more efficient
- Tasks are completed faster
- Less system resources are wasted - Command line allows you to automate repeating tasks through scripting
- Writing scripts requires you to know commands

Command Execution [edit]

- While command is being executed the shell waits for it to finish
- This state is called sleep
- When the command finishes executing the shell brings back the prompt
- It is possible to get the command prompt before the command finishes
- This requires executing a process in the background

Command Line Syntax [edit]

- A line which includes UNIX commands, program and shell script names and their arguments is called a command line
- Typical command line execution would include:
- Command line parsing
- Breaking it up into tokens
- Executing tokens
- Command line tokens are separated by whitespace

- Command line is actually executed when the Enter key is pressed

Command Editing [edit]

- Previously executed commands can be recalled
- The Bourne shell uses the up/down arrow keys to accomplish that
- Other shells may use some other mechanism, for example Korn shell uses vi-style command editing
- Recalled commands can be easily edited before re-executing
- Useful BASH keyboard shortcuts:

Go to the beginning of the line	CTRL+A
Go to the end of the line	CTRL+E
Erase Characters	Backspace or CTRL-Backspace or CTRL-h
Delete a word before the cursor	CTRL-w
Delete everything from to the beginning of line	CTRL-u
Clear Screen	CTRL-l
Search for a keyword in previous commands	CTRL+R
Auto complete file/directory names	Tab

Quoting in UNIX [edit]

- Sometimes it may be necessary to use characters that have special meaning to the shell
- In such cases such characters may need to be quoted
- There are several ways of quoting special characters in UNIX, including:

Double quotes (“ “)	quote a group of characters
Single quotes (‘ ‘)	quote a group of characters
Backslash quote (`\`)	quote the one character immediately following the backslash

- Can prevent variable substitution when the $ character is quoted

```
# shows all filenames in your pwd
$ echo *

# displays the character *
$ echo \*
```

- To quote a `\` another `\` is used, this means `\\`

' ' Quotes [edit]

- Forward single quote - different than the back tick (backward single quote)
- Quotes all that is inside, preventing wildcard and variable substitution

```
# shows all hidden files in pwd
echo .*

# displays the two characters '.' and '*'
```

```
echo '.*'
```

Double Quoting [edit]

- Commands such as echo can have their arguments quoted using double quotes
- Such quoting can preserve and/or include whitespace
- Variable substitution takes place
- Double quotes do not:
- Prevent shell variable substitution
- Stop escape characters interpretation

Categories: Pages with syntax highlighting errors | ULI101 | ULI101-2018