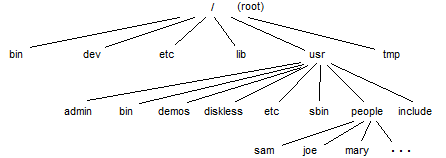
**UNIX File System**

Structure: Files are grouped into directories. A directory contains zero or more files and possibly subdirectories. This forms a hierarchical "tree" structure. The "top" directory is the "root" directory which is designated by a single /. All other directories have a name.

|  |  |
| --- | --- |
| Naming: | Names are up to 255 characters (A-Z a-z 0-9 \_ .) long. Names are case sensitive. Don't use spaces. Within a directory, all names must be unique. Filename extensions are not required, but encouraged. Any name that starts with a period is considered a "hidden" file. |
| pwd | Your "present working directory" (*pwd*) is where your shell looks for files. You are always in a specific working directory. If your shell can't find a specific file in the *pwd*, all directories in its PATH variable will be search in the order they are listed in the PATH. |
| Home ~ | The directory associated with your login account. This directory is typically the same name as a user's login name. This is a user's *pwd* at login. |
| Current **.** | A single dot represents the *pwd*. |
| Parent **..** | Two dots represent a parent directory. Every directory (except the root) has a unique "parent" directory. You don't refer to a parent directory by name because the parent directory is always unique and **..** means "go up" in the hierarchy. |
| Paths: | Accessing files not in your *pwd* requires a *path*. The path must start from 1 of 3 places:   * *pwd* path starts with **.**, **..**, or a directory name in the *pwd* * home path starts with ~ * root path starts with / |

|  |  |  |
| --- | --- | --- |
| **Task** | **Command Syntax** | **Description** |
| Where am I? | pwd | display's your "present working directory" |
| Change pwd | cd  cd *path***[[1]](#footnote-1)** | changes your *pwd* back to your home directory  changes your *pwd* to the specified directory |
| Make (create) a new directory | mkdir *directoryName* | makes (creates) a new directory with the specified name |
| Remove (delete) a directory | rmdir *directoryName* | removes (deletes) an existing **empty** directory. To delete entire file systems, use the "rm –r *DirectoryName*" command **(but with great care!)** (-r means recursive) |
| Find (search) | find *directory*   -name *filename*  -print | searches a file system for a file starting at the specified directory |
| Link in multiple locations | ln *existingFile* *newLink* | allows access to a single file from more than one directory |
| Disk usage | df  du *directoryName* | displays "disk free" information  displays "disk usage" information |

**UNIX File Globbing**

Write a pattern that matches multiple filenames. (IMPORTANT: A **glob** is **not** a regular expression!)

Notes:

* Almost all file commands allow file globbing.
* File globbing is done by the UNIX shell; different shells use different file globbing rules.
* The following discussion is for a Bash shell.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Wildcard** | **Description** | **Example** | **Matches** | **Does not match** |
| \* | matches any number of any characters including none | Law\* | Law, Laws, or Lawyer | GrokLaw, La, or aw |
| \*Law\* | Law, GrokLaw, or Lawyer. | La, or aw |
| ? | matches any single character | ?at | Cat, cat, Bat  or bat | at |
| [abc] | matches one character given in the bracket | [CB]at | Cat or Bat | cat or bat |
| [a-z] | matches one character from the range given in the bracket | Letter[0-9] | Letter0, Letter1, Letter2, up to Letter9 | Letters, Letter or Letter10 |
| [!abc] | matches one character that is not given in the bracket | [!C]at | Bat, bat, or cat | Cat |
| [!a-z] | matches one character that is not from the range given in the bracket | Letter[!3-5] | Letter1, Letter2, Letter6 up to Letter9 and  Letterx etc. | Letter3, Letter4, Letter5 or Letterxx |

Source: <https://en.wikipedia.org/wiki/Glob_(programming)>

**Brace expansion** (also called *alternation*) can be used with globbing. It generates a set of alternative combinations.

* Syntax: list the strings you want to use to create combinations in braces, { }, separated by commas.
* The braces must contain at least 2 strings; {a,} is two strings, "a" and an empty string).
* Brace expansion happens before globbing.

|  |  |
| --- | --- |
| **Examples:** | **Results:** |
| {a,b,c}x | ax bx cx |
| a{p,c,d,b}e | ape ace ade abe |
| {a,b,c}{d,e,f} | ad ae af bd be bf cd ce cf |
| \*.{jpg,jpeg,png} | \*.jpg \*.jpeg \*.png |
| \*.{png,jp{e,}g} | \*.jpg \*.jpeg \*.png |
| a{1..5} | a1 a2 a3 a4 a5 |
| file{1..4}.txt | file1.txt file2.txt file3.txt file4.txt |

Source: <https://en.wikipedia.org/wiki/Bash_(Unix_shell)#Brace_expansion>

**UNIX File Manipulation**

|  |  |
| --- | --- |
| Structure: | A UNIX file is simply a linear stream of bytes. The operating system has no knowledge of the byte organization scheme of user files. |
| Naming: | Names are up to 255 characters (A-Z a-z 0-9 \_ .) long. Names are case sensitive. Don't use spaces. Within a directory, all names must be unique. Filename extensions are not required, but encouraged. Any name that starts with a period is considered a "hidden" file. |
| Operations: | Files can be manipulated by **many** UNIX tools, some of which are listed below. |

|  |  |  |
| --- | --- | --- |
| **Task** | **Command Syntax** | **Description** |
| directory  listing | ls  ls *directory*  ls -l  ls –a  ls –R  ls –S  ls –t  ls -r | lists all of the files in your pwd  lists files in the specified directory  lists **long** information on each file  lists **all** files (including hidden files)  lists all files **recursively** (descending into sub-directories)  list files **sorted** by **size** (largest first)  list files **sorted** by modification **time** (newest first)  list files in **reverse order** |
| copy | cp *existingFile newFile* | creates a duplicate copy of a file |
| remove (delete) | rm *filename* | deletes a file - the file is gone! There is no "trash can". |
| move (rename) | mv *oldname newname* mv *filename directory* | changes the name of a file moves a file to another directory |
| display | less *filename*  cat *filename*  head *filename*  tail *filename* | display the contents of a file one "page" at a time  display the entire contents of a file to the screen  displays the first 10 lines of a file  displays the last 10 lines of a file |
| print | lp *filename* | sends a file to the "line printer" (lp) |
| combine | cat *file1 file2  file3 > resultFile* | combine the contents of multiple files |
| edit | vi *filename*  sed *filename*  awk *filename* | "visual" full screen interactive editor  "stream editor"; non-interactive  processes patterns found in files |
| compare | diff *file1* *file2*  uniq  comm *file1* *file2* | displays the differences between two files  displays lines of a file that are unique  compares sorted files |
| sort | sort *filename* | sorts the contents of a file |
| search contents | grep *expression filename*  *egrep* *expression filename fgrep* *expression filename pgrep* *expression filename* | searches the contents of a file or files for a matching expression |
| count characters | wc *filename* | counts the number of lines, words, and characters in a file |
| octal dump | od *filename* | produces an octal dump of a file (useful for non-text files) |
| spell check | spell *filename* | checks a text file for misspelled words |
| Archive (compress into a single file) | tar  ar  zip  gzip | tape archive format.  archive  package and compress (archive) files  compress files |

**UNIX File Permissions**

|  |  |
| --- | --- |
| Classes of Users: | In relationship to a particular file, all users are classified into one of three classes:  • owner the user who "owns" the file  • group a group of users, as setup by the system administrator  • other all other users not in the above two classes |
| Access Permissions: | Each file has 3 basic types of access:  • read the contents of the file can be "read" or viewed by a user  • write the contents of the file can be **changed** by a user  • execute the file can be executed by a user |
| File Access: | Every file maintains access permissions for each user class. Therefore, there are 9 access settings per file; 3 for the file owner, 3 for the group and 3 for other users. Only the owner of a file can change its access permissions. |
| Encoding: | The access permissions are displayed in a "long" directory listing (e.g., ls -l) as a series of 10 characters. These are labeled in the diagram to the right. |
| Examples: | -rwxrwxrwx 0777, can be read, changed, or executed by any user on the system.  -rwx------ 0700, can only be read, changed, or executed by its owner.  -rwx--x--x 0711, can be read, changed, or executed by its owner, and anyone else on the system can execute it.  -r--r--r-- 0444, anyone on the system can read this file, but no one can change it, not even its owner. |
| Operations: | File permissions can be manipulated by the following UNIX tools. |

|  |  |  |
| --- | --- | --- |
| **Task** | **Command Syntax** | **Description** |
| see file permissions | ls -l | (long listing) shows file permissions |
| set default  permissions | umask *bitsToDisable*  umask 0077 | sets file permissions for any new file  all new files are only accessible by their owner. |
| change permission | chmod *mask files*  chmod *who operation permission files*  who: u (user), g (group), o (other), a (all)  operation: + (add), - (remove)  permission: r (read), w (write), x (execute) | change access permissions using an octal mask.  change access permissions using symbolic operation codes. |
| change file ownership | chown *newOwner fileNames* | only the current owner (or the superuser) can change the ownership of a file |
| change file group | chgrp *newGroup fileNames* | only the current owner (or the superuser) can change the group membership of a file |
| get file details | stat filename | displays details about a file or directory |

1. Remember, the bash shell does auto-completion on file names. Use the *tab key* to complete filenames. [↑](#footnote-ref-1)