FREE eBook

LEARNING unix

Free unaffiliated eBook created from
Stack Overflow contributors.

#unix

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: unix

It is an unofficial and free unix ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official unix.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with unix

Remarks

This section provides an overview of what unix is, and why a developer might want to use it.

It should also mention any large subjects within unix, and link out to the related topics. Since the Documentation for unix is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting unix set up or installed.

Read Getting started with unix online: https://riptutorial.com/unix/topic/912/getting-started-with-unix

Chapter 2: Basic console commands

Examples

pwd - print working directory

```
$> pwd
/home/myUserHome
$> cd ..
$> pwd
/home
```

will print the current path to the console.

pushd/popd (store current dir on stack and go to dest / pop prev dir and go to it)

```
$ pwd
/home/bob/somedir1/somedir2/somedir3

$ pushd /home/bob/otherdir1/otherdir2
/home/bob/otherdir1/otherdir2 /home/bob/somedir1/somedir2/somedir3

$ popd
/home/bob/somedir1/somedir2/somedir3

$ pushd /usr
/usr /home/bob/somedir1/somedir2/somedir3

$ pushd /var
/var /usr /home/bob/somedir1/somedir2/somedir3

$ popd
/usr /home/bob/somedir1/somedir2/somedir3

$ pwd
/usr

$ popd
/home/bob/somedir1/somedir2/somedir3

$ pwd
/home/bob/somedir1/somedir2/somedir3
```

file manipulation commands

List of commands that will be introduced here:

```
ls     #view contents of a directory
touch  #create new file
mkdir  #create new directory
cp     #copy contents of one file to another
```

```
mv     #move file from one location to another
rm     #delete a file or directory
```

ls examples

```
jennifer@my_computer:~/Desktop$ ls
c++ projects    Research Paper.docx     test.cpp
```

shows the current directory

```
jennifer@my_computer:~/Desktop$ ls c++\ projects
DNA_analysis.cpp        encryption.cpp  pool_game.cpp
```

shows the directory "c++ projects". Space characters in file names are typed as "\ ".

touch example

```
jennifer@my_computer:~/Desktop$ ls
c++ projects    Research Paper.docx     test.cpp
jennifer@my_computer:~/Desktop$ touch ruby_test.rb
jennifer@my_computer:~/Desktop$ ls
c++ projects    Research Paper.docx     ruby_test.rb    test.cpp
```

mkdir example

```
jennifer@my_computer:~/Desktop$ mkdir ruby
jennifer@my_computer:~/Desktop$ ls
c++ projects    Research Paper.docx     ruby    ruby_test.rb    test.cpp
jennifer@my_computer:~/Desktop$ cd ruby
jennifer@my_computer:~/Desktop/ruby$ ls
<nothing>
jennifer@my_computer:~/Desktop/ruby
```

It doesn't actually print `<nothing>`. It's just how I'm representing that it doesn't output anything

cp examples

```
jennifer@my_computer:~/Desktop/ruby$ cd ..
jennifer@my_computer:~/Desktop$ cp test.cpp c++_test.cpp
jennifer@my_computer:~/Desktop$ ls
c++ projects    c++_test.cpp    Research Paper.docx     ruby    ruby_test.rb
test.cpp
```

This is when the last arg to `cp`, in this case "c++_test.cpp" is not an existing directory. `cp` will create a file called "c++_test.cpp", with contents identical to that of "test.cpp". If c++_test.cpp already existed, `cp` would have deleted what was previously there before copying the contents of "test.cpp" over.

```
jennifer@my_comptuer:~/Desktop$ ls ruby
<nothing>
jennifer@my_computer:~/Desktop$ cp ruby_test.rb ruby
jennifer@my_computer:~/Desktop$ ls ruby
```

```
ruby_test.rb
```

This is what happens when the last arg to `cp`, in this case "ruby", is a directory. `cp` creates a file with the same name as "ruby_test.rb", but in the directory "ruby".

mv examples

```
jennifer@my_computer:~/Desktop$ ls
c++ projects    c++_test.cpp    Research Paper.docx     ruby    ruby_test.rb
test.cpp
jennifer@my_computer:~/Desktop$ mv ruby_test.rb ruby\ test.rb
jennifer@my_computer:~/Desktop$ ls
c++ projects    c++_test.cpp    Research Paper.docx     ruby    ruby test.rb
test.cpp
```

This is what happens when the last arg to `mv`, in this case "ruby test.rb", is not an existing directory. The file "ruby_test.rb" has been renamed to "ruby test.rb". If "ruby test.rb" already existed, it would have been overwritten Note, again, that spaces are preceded by a \.

```
jennifer@my_computer:~/Desktop$ ls
c++ projects    c++_test.cpp    Research Paper.docx     ruby    ruby test.rb
test.cpp
jennifer@my_computer:~/Desktop$ ls c++\ projects
DNA_analysis.cpp        encryption.cpp  pool_game.cpp
jennifer@my_computer:~/Desktop$ mv test.cpp c++\ projects
jennifer@my_computer:~/Desktop$ ls
c++ projects    c++_test.cpp    Research Paper.docx     ruby    ruby test.rb
jennifer@my_computer:~/Desktop$ ls c++\ projects
DNA_analysis.cpp        encryption.cpp  pool_game.cpp   test.cpp
```

This is what happens when `mv` is a directory that already existed. The file "test.cpp" gets moved to the directory "c++ projects".

rm examples

```
jennifer@my_computer:~/Desktop$ ls
c++ projects    c++_test.cpp    Research Paper.docx     ruby    ruby test.rb
jennifer@my_computer:~/Desktop$ rm c++_test.cpp
jennifer@my_computer:~/Desktop$ ls
c++ projects    Research Paper.docx     ruby    ruby test.rb
```

c++_test.cpp has been deleted

```
jennifer@my_computer:~/Desktop$ rm c++\ projects
rm: cannot remove 'c++ projects': Is a directory
jennifer@my_computer:~/Desktop$ ls
c++ projects    Research Paper.docx     ruby    ruby test.rb
```

`rm` has an extra requirement to delete directories

```
jennifer@my_computer:~/Desktop$ rm -rf c++\ projects
jennifer@my_computer:~/Desktop$ ls
Research Paper.docx     ruby    ruby test.rb
```

`-rf` must be added to delete a directory.

To learn more about `ls`, type the command `ls --help`. For `touch`, type `touch --help`. Likewise with all 6 commands mentioned here. This prints out a detailed explanation of use without creating or deleting anything.

cd command, directories explained

```
michael@who-cares:~$
```

The symbol `~` after the `who-cares:` is the current directory. `~` actually means the person's home directory. In this case, that's `/home/michael`.

```
michael@who-cares:~$ cd Downloads
michael@who-cares:~/Downloads$
```

Looks for `Downloads` in the current directory, then makes that the current directory.

```
michael@who-cares:~/Downlaods$ cd /var
michael@who-cares:/var$
```

Since this directory started with a `/`, that means look in the root directory for the directory `var`. For those coming from windows, the root directory is the equivalent to `C:\`. Directories starting with `/` are called **"absolute directories"** and directories that don't are called **"relative directories"**

```
michael@who-cares:/var cd lib/dbus
michael@who-cares:/var/lib/dbus$
```

The `/` in the middle means do `cd lib` and once that's done `cd dbus` in one command.

```
michael@who-cares:/var/lib/dbus$ cd .
michael@who-cares:/var/lib/dbus$
```

`.` actually means "the current directory". The command `cd .` is basically useless, but `.` is useful for other things.

```
michael@who-cares:/var/lib/dbus$ cd ..
michael@who-cares:/var/lib$
```

`..` actually means "the parent of the current directory". As such, `cd ..` means "navigate one directory up".

```
michael@who-cares:/var/lib$ cd ../log/apt
michael@who-cares:/var/log/apt$
```

`.` and `..` can also be part of the `/` chain. Also, there's no limit to how long it can be.

```
michael@who-cares:/var/log/apt$ cd /dev/bus
```

```
michael@who-cares:/dev/bus$
```

The `/` chain can even exist when the directory starts at root.

```
michael@who-cares:/dev/bus$ cd /
michael@who-cares:/$
```

`cd /` takes you to the root directory. I wonder what happens if you type `cd ..` here... (don't worry. It's safe)

```
michael@who-cares:/$ cd home
michael@who-cares:/home$ cd michael
michael@who-cares:~$
```

Every user has a directory for their stuff inside the home directory. If current directory is under the home directory, that part of the name, in this case `/home/michael`, it's replaced with `~`.

```
michael@who-cares:~$ cd sys
michael@who-cares:/sys$ cd ~/Desktop
michael@who-cares:~/Desktop$ cd ~/..
michael@who-cares:/home$
```

`~` can also be part of the `/` chain. It can even be in the same chain as `...` If the directory starts with `~`, it's an absolute directory just like if it starts with `/`.

Last thing to try: type `cd` with no directory after.

which

To determine where on your system an executable in your path exists, use the `which` command:

```
$ which python
$
```

If there is no response, that executable does not exist in your path. The system will simply return you a new prompt without an error message. If the executable does exist on your path, it will show the directory where it actually exists:

```
$ which ls
/bin/ls
```

This can be helpful in determining why behavior does not match expectation by ensuring you're executing the version of the executable that you think you are. For instance, if you have both Python 2 and Python 3 installed, they both might be executed by typing `python` in the terminal - but the executable actually being run may be different than expected. As shown above this command will work for any standard unix command, which are all backed by individual executables.

Basic Unix commands

```
$pwd
```

Displays the present working directory.

```
$who
```

Displays all the users logged in.

```
$who am i
```

Shows the username of the current user.

```
$date
```

Displays the current system date

```
$which <command>
```

Shows the path of the specified command. For example "$which pwd" will shows the path of 'pwd' command.

```
$file <file_name>
```

Shows the type of the specified file(regular file, directory or other files)

```
$cal
```

Displays the calender of the current month.

```
$bc
```

Shows the mathematical calculation between two integers of floats. For example "$bc 2+3" will returns the arithmetic sum of 3 and 5.

```
$ls
```

Lists the contents of the directory.

- $ls -l : lists in long format.
- $ls -c : Multi column output.
- $ls -f : Lists the type of file.
- $ls -r : Recursive listing of all subdirectories encountered.
- $ls -a : Displays all files including hidden files.
- $ls -i : Lists all files along with its I-Node number.

```
$grep [options] <pattern> <input_file_names>
```

Prints lines that contain a match for a pattern.

- $grep -i : Perform case insensitive matching
- $grep -v : Prints all lines that don’t contain the regex
- $grep -r : Recursively search subdirectories listed and prints file names with occurrence of the pattern
- $grep -I : Exclude binary files

Read Basic console commands online: https://riptutorial.com/unix/topic/4262/basic-console-commands

Chapter 3: Getting Started with Unix Commands

Introduction

This topic will provide a comprehensive coverage of basic Unix commands.

Examples

A non exhaustive list of Unix commands

```
$man <command>
```

Displays the on-line manual pages for the command

```
$clear
```

Clears the terminal screen

```
$pwd
```

Returns the working directory name

```
$echo <string>
```

Writes the string to the standard output

```
$printf <string>
```

Format and print the string Example: print $PATH $printf “%s\n” $PATH

```
$uptime
```

Show how long system has been running

```
$which <program>
```

Locate a program file in the user’s path

```
$whereis <program>
```

Checks the standard binary directories for the specified programs, printing out the

FILES

```
$cd [directory]
```

Change directory Commonly used directory symbols:

- . : Current directory
- .. : Parent directory
- ~ : Home directory
- / : Root directory

```
$ls
```

- $ls -a : Show hidden files
- $ls -l : Show long list
- $ls -1 : Show just the filename per line
- $ls -h : Human readable format

  $file Determine file type (e.g. gzip)

READING FILES

```
$more
```

Display content of a file one screen at a time
spacebar : Scroll to next screen; b=previous screen
enter : Scroll one line
h : Help for more
q : Quit help

```
$less <file>
```

Less is a program similar to more, but which allows backward movement in the file as well as forward movement

```
$cat <file>
```

Reads files sequentially, writing them to the standard output

```
$head [-number] <file>
```

Display first lines of a file

```
$tail [-number] <file>
```

Displays the contents of file or, by default, its standard input, to the standard output

- $tail -f : View changes in file in real-time

```
$touch <file>
```

Sets the modification and access times of files. If any file does not exist, it is created with default permissions

```
$tee <file>
```

Copies standard input to standard output. Press ctrl-d to stop adding content

- $tee -a : Append the output to the files rather than overwriting them

```
$mkdir <directory>
```

Create a directory

```
$wc
```

Display the number of lines, words, and bytes contained in each input file, or standard input

- wc -l : Count lines
- wc -w : Count words
- wc -m : Count characters

```
$diff <file1> <file2>
```

Compare two files line by line. Will print only the different lines.

```
$locate <file>
```

Locate files on disk

- $locate -q : suppress errors

```
$find <path> <expression> <action>
```

Search for files by name or content

- $find -name : Find by filename
- $find -size <+/-n> Example: Find files in current directory that are larger than 10k $find . -size +10

```
$rm <file or directory>
```

Delete <file or directory>

- rm -f : Skip confirmation
- rm -i : Approve each deletion
- rm -r : Recursive

Example: Delete the and it’s content $rm -r

```
$mv <source_file> <target_file>
```

Renames a file

```
$mv <source_file> <target_directory>
```

Moves a file

- $mv -i : Don’t override exiting files
- $mv -r : Recursive

Example: move directory up in hierarchy $mv ..

```
$cp
```

Copy a file/directory within the same machine (Use scp command to copy to a remote machine)

- $cp -i : Don’t override exiting files
- $cp -r :recursive

Example: Copy and rename a file

```
$cp <file_name> <new_file_name>
```

Example: Copy to directory

```
$cp <file_name> <directory_name>
```

Example: Copy and rename a directory

```
$cp –R <directory> <new_directory>
```

Example: Copy all files of specific type to a directory

```
$cp *.txt <directory>
```

```
$ln –s <file> <link name>
```

Create an alias (link) to a file

- $ln -s : Create a soft link (A link that functionas across machines)

```
$sort <file>
```

Sort the content of a file -r reverse sorts -n numeric sort

Example: sort the and write the result to sorted.txt $sort | uniq -u > sorted.txt

```
$uniq [-ucd] filename(s)
```

Looks for duplicate lines. Data must be sorted first

- $uniq -d : show only one copy of the duplicate lines
- $uniq -u : Show only lines that are not duplicate
- $uniq -c : Output each line preceded by a count of occurrences Example: show users that are connected more than once $who | cut -d' ' -f1 | sort | uniq -d

```
$grep <pattern> <file_name>
```

Prints lines that contain a match for a pattern.

- $grep -i : Perform case insensitive matching
- $grep -v : Prints all lines that don't contain the regex
- $grep -r : Recursively search subdirectories listed and prints file names with occurrence of the pattern
- $grep -l : Exclude binary files

```
$tr "string1" ["string 2"]
```

Search and replace tool. tr only accepts its input from pipes and redirections. it doesn't accept files as input.

- tr -d : Delete all occurrences of all CHARACTERS in string1

Example: Print a.txt to screen after deleting all occurences of ";" $cat a.txt | tr -d ";"

- tr -s : Replace occurrences with a single character

Example: $echo "SSSS SS" | tr -s "S" "S"

```
$tar
```

Creates and manipulates streaming archive files. This implementation can extract from tar, pax, cpio, zip, jar, ar, and ISO images and can create tar, pax, cpio, ar, and shar archives.

DISK USAGE

```
$du [file or directory]
```

Display the file system block usage for each file or directory. If no file/directory is specified, the block usage of the current directory is displayed.

- $du -a : Files & directories (default is directories only)
- $du -h : Human readable format Example: Display disk usage, ordered, only MB files $du -h | grep -i “m\t” | sort -n Example: Find out top 10 largest file/directories $du -a /var | sort -n -r | head -n 10

```
$df
```

Display free disk space

- $df -h : Human readable format

REDIRECTIONS & PIPES

> Redirect standard output. Dont overwrite file if it exists

>! Redirect standard output. Overwrite file if it exists

>& Redirect standard output and standard error

Example: Redirect command output into a file

```
$ls > result.txt
```

Use > /dev/null file to dispose of errors message

Example: Find a file named my_file_name and print the result to ~/find.txt; Hide errors (e.g. “permissions denied”)

```
$find / -name my_file_name.* > /dev/null > ~/find.txt
```

< Redirect standard input

>> Append standard output <command> >> : Append output to the end of an existing

<command> < : Redirect input to a command from a file

| Redirect standard output to another command (pipe)

Example: Show paginated details of running processes

```
$ps -ex | more
```

| : Pipe output of command1 to be the input of command2 (If an output file is desired in the middle of a pipe use the tee command)

Example: Count the number of connected users

```
$who | wc -l
```

PROCESSES

```
$ps
```

Show active processes

- $ps -e : Show information about the process
- $ps -x : Show hidden processes

Example: Find processes by name $grep -l <process_name_regex>

```
$kill [-signal] pid
```

Kill a process.

Some of the more commonly used signals:

- 3 : QUIT (quit)
- 9 : KILL (non-catchable, non-ignorable kill)
- 15 : TERM (software termination signal)

```
$top
```

Display and update sorted information about processes See man pages for list of possible keys. common keys are: cpu, threads, ports

- $top -o

```
$htop
```

Display and update sorted information about processes

USER & PERMISSIONS

```
$sudo <command>
```

Execute the command as a super user

```
$su
```

(substitute user) opens a session as an admin

```
$exit
```

Exit root

```
$whoami
```

Display effective user id

```
$who
```

Print all connected user names

```
$passwd
```

Change password

```
$chmod <who> <operation> <permissions> <file or directory name>
```

Change owner/group access to a file or directory

Who: u user; g group; o other; a all above

Operation: + add; - remove; = set (meaning reset to nothing and set only what was specified)

Permissions: r w x

Example: Adds read/execute permissions to group

```
$chmod g +rx <file>
```

Example:

```
$chmod 743 <file>
```

Note that to $cd into a directory you need the x permissions

Read Getting Started with Unix Commands online: https://riptutorial.com/unix/topic/9848/getting-started-with-unix-commands

Chapter 4: Overview of Unix

Introduction

Developed in AT&T Bell Labs by Ken Thomson as a single user OS in 1969. Initially written in assembly language Developed as multi-user OS. later rewritten in C in 1973 Licensed to university for educational purposes in 1974 POSIX (Portable Operating System for Unix) was developed

Examples

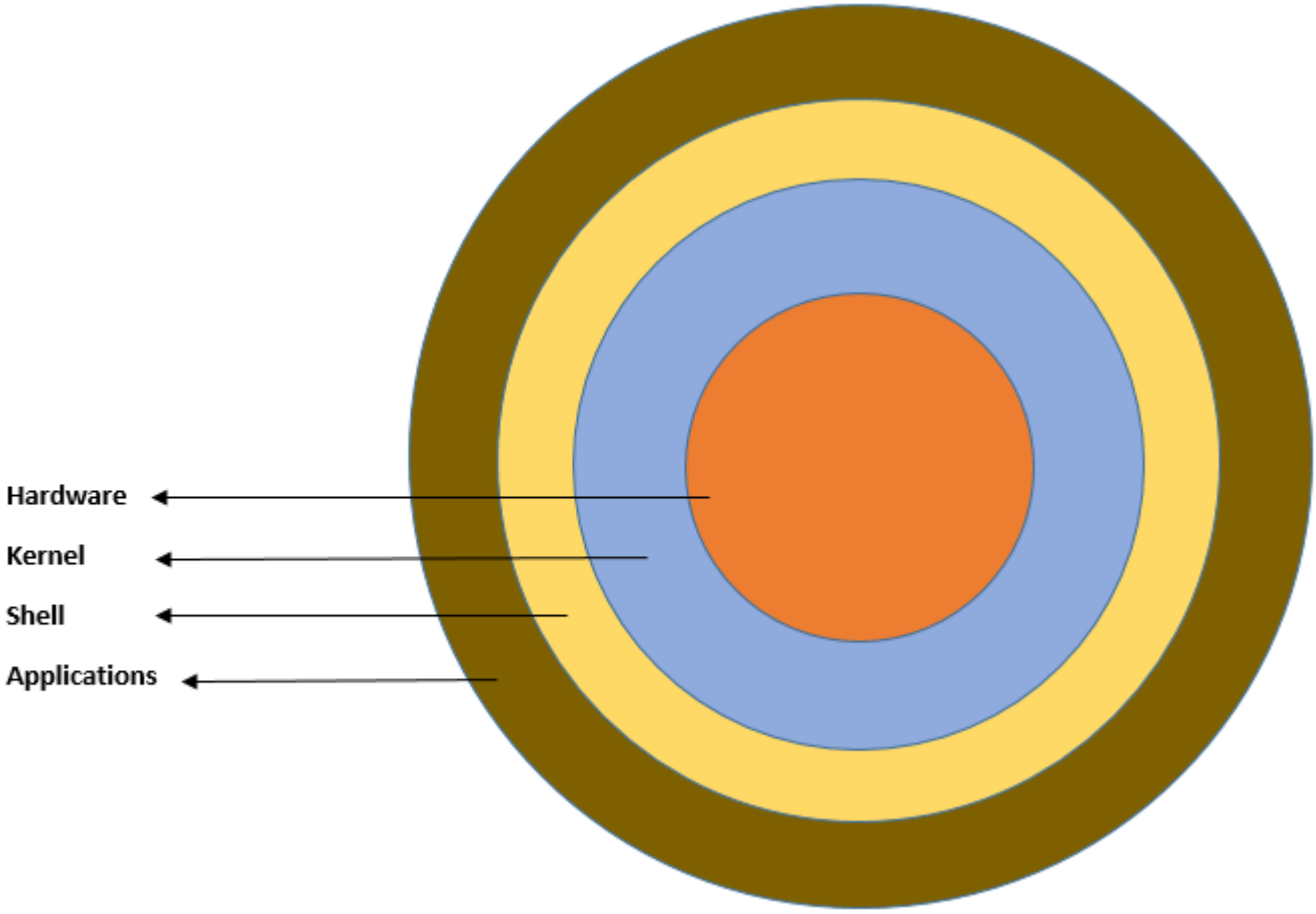
Unix Flavours

- AIX by IBM
- Solaris by Sun Microsystems
- HP-UX by Hewlett Packard
- IRIX by Silicon Graphics, Inc
- FreeBSD by Free BSD Group
- GNU/Linux by Open Source Movement
- SCO Unix by The Santa Cruz Operation Inc

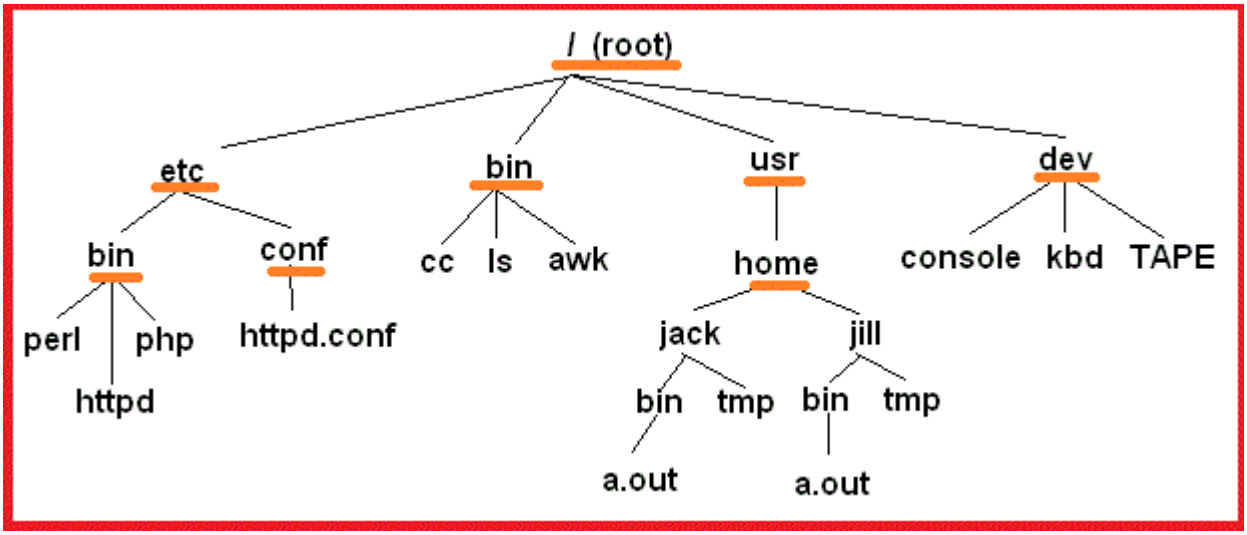
Features of UNIX

- Multi user
- Multi tasking
- Interactive
- Shell
- Security
- Hierarchial file system

Unix Architecture



Unix File Systems



Read Overview of Unix online: https://riptutorial.com/unix/topic/9338/overview-of-unix

Chapter 5: Permissions

Introduction

In unix each files has certain permissions like read, write and execute. A user can manipulate the permissions of a file using 'chmod' command.

Remarks

In UNIX, there are three permissions used to grant a certain level of access to a file or folder.

For files:

- **Read**: Allow the user/group/others to read a file.
- **Write**: Allow the user/group/others to modify a file.
- **Execute**: Allow the user/group/others to execute (or run) a file.

These are slightly changed for directories:

- **Read**: Allow the user/group/others to list the names of files in a directory.
- **Write**: Allow the user/group/others to create, delete and rename files in a directory.
- **Execute**: Allow the user/group/others to access file metadata and contents for a directory.

These permissions can be represented using the letters "r" for read, "w" for write, and "x" for execute. They can also be represented numerically: 4 for read, 2 for write and 1 for execute.

Examples

Change a file's permissions

```
> chmod 644 example.txt
> ls -l example.txt
-rw-r--r--  1 owner ogroup 57 Jul  3 10:13  example.txt
```

The above command changes the file permissions to allow the file owner to read and write to a file. It also allows users in the owner's group and other users in the system to read the file.

Understanding Permissions

Let's say there is a file we would like to execute, a bash script named `add.sh`, for example. Typing `./add.sh` however, yields a permission error. Getting the permissions is a simple process.

To determine the permissions a file has, type:

`ls -l filename`, or, in our case, `ls -l ./add.sh`

This prints the following to the console:

```
-r--r--r-- 1 username groupname 0 Jan 4 12:00 add.sh
```

Let's stop and understand what this means. There are three different types for permissions: owner, group, others. Distinct permissions apply to each permission type.

There are also three permission actions, which more broadly also describe what exactly a user can do to a file. These are: (read: r, write: w, execute: x).

So, back up to that string of dashes and r's. Each permission group has three potential abilities. The groups are listed in the order `owner-group-others` and the actions as `read-write-execute`.

But wait, that means there's an extra character at the beginning of the string. This is actually the file descriptor character. We can see there is a `-` there, but other characters exist for things like directories `(d)`, sockets `(s)`, symbolic link `(l)` etc.

This leaves us with essentially this information: `a file where owner, group, and others have read permissions. No other permissions granted.`

Let's alter this to allow the owner to also write and execute the file. Note: Depending on the permissions, it may be necessary to prepend `sudo` to this command.

```
chmod 744 add.sh
ls -l add.sh
```

Prints out

```
-rwxr--r-- 1 username groupname 0 Month time add.sh
```

Now, the owner of the file can execute the file by typing

```
./add.sh
```

CHMOD calculation

CHMOD Calculation

CHMOD is binary. _ / _ _ _ / _ _ _ / _ _ _ = _/4+2+1/4+2+1/4+2+1 = 777 = _/rwx/rwx/rwx = 777
Therefore _rwx = _/4+2+1 = 7

D / _ _ _ / _ _ _ / _ _ _ (‘D’ = directory, another use is L = Link)

So e.g. _rwxr_xr_x = _/rwx/rx/r_x = 755

CHOWN

To change own:group you use command chown user:group

e.g. chown owner:group or if owner and group are same you can use chown owner: (because

linus assumes owner:group are same).

Read Permissions online: https://riptutorial.com/unix/topic/6394/permissions

Chapter 6: View the Manual Pages

Examples

Viewing the Manual Page for a System Command

```
man <command>
```

This will show the manual page for the specified command.

For example, `man ping` will show:

```
PING(8)                   BSD System Manager's Manual                  PING(8)

NAME
     ping -- send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS
     ping [-AaCDdfnoQqRrv] [-b boundif] [-c count] [-G sweepmaxsize]
          [-g sweepminsize] [-h sweepincrsize] [-i wait] [-k trafficclass]
          [-l preload] [-M mask | time] [-m ttl] [-P policy] [-p pattern]
          [-S src_addr] [-s packetsize] [-t timeout] [-W waittime] [-z tos]
          host
     ping [-AaDdfLnoQqRrv] [-b boundif] [-c count] [-I iface] [-i wait]
          [-k trafficclass] [-l preload] [-M mask | time] [-m ttl] [-P policy]
          [-p pattern] [-S src_addr] [-s packetsize] [-T ttl] [-t timeout]
          [-W waittime] [-z tos] mcast-group

DESCRIPTION
     The ping utility uses the ICMP protocol's mandatory ECHO_REQUEST datagram
     to elicit an ICMP ECHO_RESPONSE from a host or gateway.  ECHO_REQUEST
     datagrams (``pings'') have an IP and ICMP header, followed by a ``struct
     timeval'' and then an arbitrary number of ``pad'' bytes used to fill out
     the packet.  The options are as follows:

...
```

While viewing the manpage it can be searched. Typing a slash (`/`) followed by the search term will jump to the first occurence of the term. Example: `/ping`

Pressing `N` afterwards will skip to the next occurrence. `Shift+N` will jump to the previous ocurrence.

Get the File Path for a Manual Page

```
$ man -w find
/usr/share/man/man1/find.1.gz

$ man -w printf
/usr/share/man/man1/printf.1.gz

$ man -w man
/usr/share/man/man1/man.1.gz
```

Search for a Manual Page

You can search for `man` pages containing a particular string in their description using:

```
man -k <string>
```

For example:

```
man -k unzip
```

Might return:

```
man -k unzip
IO::Uncompress::Bunzip2(3pm) - Read bzip2 files/buffers
IO::Uncompress::Gunzip(3pm) - Read RFC 1952 files/buffers
IO::Uncompress::Unzip(3pm) - Read zip files/buffers
PerlIO::gzip(3pm)        - Perl extension to provide a PerlIO layer to gzip/gunzip
gzip(1), gunzip(1), zcat(1) - compress or expand files
IO::Uncompress::Bunzip2(3pm) - Read bzip2 files/buffers
IO::Uncompress::Gunzip(3pm) - Read RFC 1952 files/buffers
IO::Uncompress::Unzip(3pm) - Read zip files/buffers
PerlIO::gzip(3pm)        - Perl extension to provide a PerlIO layer to gzip/gunzip
bzip2(1), bunzip2(1)     - a block-sorting file compressor, v1.0.6 bzcat -
decompresses files to stdout bzip2recover - recovers data from damaged bzip2 files
funzip(1)                - filter for extracting from a ZIP archive in a pipe
unzip(1)                 - list, test and extract compressed files in a ZIP archive
unzipsfx(1)              - self-extracting stub for prepending to ZIP archives
```

Find a man page in a different section

Sometimes a term is defined in multiple sections of the manual. By default, `man` will only display the first page it finds, which can be annoying for programmers because C functions are documented in a later section than commands and system calls. Use the following to display all pages that match a name:

```
$ man -wa printf
/usr/share/man/man1/printf.1.gz
/usr/share/man/man1p/printf.1p.gz
/usr/share/man/man3/printf.3.gz
/usr/share/man/man3p/printf.3p.gz
```

To view the page from a specific section, simply place it before the term:

```
man 3 printf
```

Read a manual file with man

This is same as reading a manual for a command:

```
man /path/to/man/file
```

Read View the Manual Pages online: https://riptutorial.com/unix/topic/442/view-the-manual-pages

Credits

S. No	Chapters	Contributors
1	Getting started with unix	Community
2	Basic console commands	Anand C, Boon, Dirk Schumacher, eli-bd, Jean-Baptiste Yunès, Nathaniel Ford, SarcasticSully
3	Getting Started with Unix Commands	eli-bd
4	Overview of Unix	Anand C
5	Permissions	Anand C, Chris Forrence, Joey Chatterjee, rama chandra sunkara, William Carron
6	View the Manual Pages	Benjamin W., drunken_monkey, intboolstring, Jahid, Simon Jester, Srinidhi, Will