AUTHOR:
        Hashrat and libUseful are (C) 2014 Colum Paget. They are released under the Gnu Public License so you may do anything with them that the GPL allows.

Email: colums.projects@gmail.com
Blog: http://idratherhack.blogspot.com

DISCLAIMER:
  This is free software. It comes with no guarentees and I take no responsiblity if it makes your computer explode or opens a portal to the demon dimensions, or does anything at all, or doesn't.

SYNOPSIS:
        Hashrat is a hash-generation utility that supports the md5, sha1, sha256, sha512, whirlpool, jh-244, jh256, jh-384 and jh-512 hash functions, and also the HMAC versions of those functions. It can output in 'traditional' format (same as md5sum and shasum and the like), or it's own format. Hashes can be output in octal, decimal, hexadecimal, uppercase hexadecimal or base64. It supports directory recursion, hashing entire devices, and generating a hash for an entire directory. It has a 'CGI' mode that can be used as a web-page to lookup hashes.

CREDITS:
        Thanks for bug reports/advice to: Stephan Hegel, Michael Shigorin <mike@altlinux.org> and Joao Eriberto Mota Filho <eriberto@debian.org>
        Thanks to the people who invented the hash functions!
        MD5: Ronald Rivest
        Whirlpool: Vincent Rijmen, Paulo S. L. M. Barreto
        JH: Hongjun Wu
        SHA: The NSA (thanks, but please stop reading my email. It's kinda creepy.).

        Special thanks to Professor Hongjun Wu for taking the time to confirm that his JH algorithm is free for use in GPL programs.
        Special, special thanks to Joao Eriberto Mota Filho for doing a LOT of work to make hashrat debian ready!

USAGE:
```
    hashrat [options] [paths]...


                Hash things:            hashrat [options] [paths to hash]
                Check hashes:           hashrat -c [options] [paths to hash]
                Find files matching:    hashrat -m [options] [paths to hash]
                Find duplicate files:           hashrat -dups [options] [paths
to hash]
```

Options:
```
  --help          Print this help
  -help           Print this help
  -?              Print this help
  --version       Print program version
  -version        Print program version
        -type <type>    Use hash algorithmn <type>. Types can be chained
together as a comma-seperated list.
  -md5            Use md5 hash algorithmn
  -sha1           Use sha1 hash algorithmn
  -sha256         Use sha256 hash algorithmn
  -sha512         Use sha512 hash algorithmn
  -whirl          Use whirlpool hash algorithmn
  -whirlpool      Use whirlpool hash algorithmn
  -jh224          Use jh-224 hash algorithmn
  -jh256          Use jh-256 hash algorithmn
  -jh384          Use jh-384 hash algorithmn
  -jh512          Use jh-512 hash algorithmn
  -hmac           HMAC using specified hash algorithm
  -8              Encode with octal instead of hex
  -10             Encode with decimal instead of hex
  -H              Encode with UPPERCASE hexadecimal
  -HEX            Encode with UPPERCASE hexadecimal
  -64             Encode with base64 instead of hex
  -base64         Encode with base64 instead of hex
  -i64            Encode with base64 with rearranged characters
  -p64            Encode with base64 with a-z,A-Z and _-, for best compatibility
with 'allowed characters' in websites.
  -x64            Encode with XXencode style base64.
  -u64            Encode with UUencode style base64.
  -g64            Encode with GEDCOM style base64.
  -a85            Encode with ASCII85.
  -z85            Encode with ZEROMQ variant of ASCII85.
  -t              Output hashes in traditional md5sum, shaXsum format
  -trad           Output hashes in traditional md5sum, shaXsum format
  -bsd            Output hashes in bsdsum format
  -tag            Output hashes in bsdsum format
  --tag           Output hashes in bsdsum format
  -r              Recurse into directories when hashing files
  -f <listfile>   Hash files listed in <listfile>
  -i <pattern>    Only hash items matching <pattern>
  -x <pattern>    Exclude items matching <pattern>
  -n <length>     Truncate hashes to <length> bytes
  -c              CHECK hashes against list from file (or stdin)
  -cf             CHECK hashes but only show failures
  -C              CHECK files against list from file (or stdin) can spot new
files
  -Cf             CHECK files but only show failures
  -m              MATCH files from a list read from stdin.
  -lm             Read hashes from stdin, upload them to a memcached server
(requires the -memcached option).
  -X              In CHECK or MATCH mode only examine executable files.
  -exec           In CHECK or MATCH mode only examine executable files.
  -dups           Search for duplicate files.
  -memcached <server> Specify memcached server. (Overrides reading list from
stdin if used with -m, -c or -cf).
  -mcd <server>   Specify memcached server. (Overrides reading list from stdin
if used with -m, -c or -cf).
  -h <script>     Script to run when a file fails CHECK mode, or is found in
MATCH mode.
  -hook <script>  Script to run when a file fails CHECK mode, or is found in
FIND mode
  -color          Use ANSI color codes on output when checking hashes.
  -strict         Strict mode: when checking, check file mtime, owner, group,
and inode as well as it's hash
  -S              Strict mode: when checking, check file mtime, owner, group,
and inode as well as it's hash
  -d              dereference (follow) symlinks
  -fs             Stay on one file system
  -dirmode        DirMode: Read all files in directory and create one hash for
them!
  -devmode        DevMode: read from a file EVEN OF IT'S A DEVNODE
  -lines          Read lines from stdin and hash each line independantly.
  -rawlines       Read lines from stdin and hash each line independantly,
INCLUDING any trailing whitespace. (This is compatible with 'echo text |
md5sum')
```

```
  -rl             Read lines from stdin and hash each line independantly,
INCLUDING any trailing whitespace. (This is compatible with 'echo text |
md5sum')
  -cgi            Run in HTTP CGI mode
  -net            Treat 'file' arguments as either ssh or http URLs, and pull
files over the network and then hash them (Allows hashing of files on remote
machines).
                  URLs are in the format ssh://[username]:[password]@[host]:
[port] or http://[username]:[password]@[host]:[port]..
  -idfile <path>  Path to an ssh private key file to use to authenticate INSTEAD
OF A PASSWORD when pulling files via ssh.
  -xattr          Use eXtended file ATTRibutes. In hash mode, store hashes in
the file attributes, in check mode compare against hashes stored in file
attributes.
  -txattr         Use TRUSTED eXtended file ATTRibutes. In hash mode, store
hashes in 'trusted' file attributes. 'trusted' attributes can only be read and
written by root.
  -attrs          comma-separated list of filesystem attribute names to be set
to the value of the hash.
  -cache          Use hashes stored in 'user' xattr if they're younger than the
mtime of the file. This speeds up outputting hashes.
  -u <types>      Update. In checking mode, update hashes for the files as you
go. <types> is a comma-separated list of things to update, which can be 'xattr'
'memcached' or a file name. This will update these targets with the hash that
was found at the time of checking.
  -hide-input     When reading data from stdin in linemode, set the terminal to
not echo characters, thus hiding typed input.
  -star-input     When reading data from stdin in linemode replace characters
with stars.
```

Hashrat can also detect if it's being run under any of the following names
(e.g., via symlinks)

```
  md5sum          run with '-trad -md5'
  shasum          run with '-trad -sha1'
  sha1sum         run with '-trad -sha1'
  sha256sum       run with '-trad -sha256'
  sha512sum       run with '-trad -sha512'
  jh224sum        run with '-trad -jh224'
  jh256sum        run with '-trad -jh256'
  jh384sum        run with '-trad -jh384'
  jh512sum        run with '-trad -jh512'
  whirlpoolsum    run with '-trad -whirl'
  hashrat.cgi     run in web-enabled 'cgi mode'
```

USE EXAMPLES:

```
        hashrat

                Generate an md5 hash of data read from stdin  (default hash type
is md5).

        hashrat -jh256

                Generate a jh-256 hash of data read from stdin

        hashrat -sha256 -64

                Generate an sha-256 hash of data read from stdin, output with
base64 encoding.
```

```
	hashrat -sha256 -64 -lines
```

Read lines from stdin, and generate an sha-256 with base64 encoding FOR EVERY LINE. This strips any whitespace from the end of the line (including \r and/or \n line terminators).

```
	hashrat -md5 -trad -rawlines
```

Read lines from stdin, and generate an md5 hash in 'traditional' format for every line INCLUDING TRAILING WHITESPACE. This is compatible with 'echo text | md5sum' where 'text' is one line, as 'echo' adds a newline to the end of the text it outputs.

```
	hashrat -type sha256,whirl,md5
```

Read data from stdin, hash it with sha256, then hash the resulting hash with whirlpool, then with md5

```
	hashrat *
```

Generate a list of hashes for files in the current directory (default hash type is md5).

```
	hashrat -r -sha1 * > hashes.sha1
```

Generate a list of hashes for files in the current directory, AND ALL SUBDIRECTORIES, using sha1 hashing.

```
	cat hashes.sha1 > hashrat -c
```

Check hashes listed in 'hashes.sha1'

```
	cat hashes.sha1 > hashrat -c -strict
```

Check hashes listed in 'hashes.sha1'. If hashes are NOT in 'traditional' format than the '-strict' flag will cause hashrat to check the files uid, gid, size, mtime and inode and print a failure message if any of those don't match.

```
	cat hashes.sha1 > hashrat -cf
```

Check hashes listed in 'hashes.sha1' but only output failures

```
	cat APT1.md5 | hashrat -m -r /
```

Read a list of hashes from stdin, and search recursively for files matching them.

```
	cat APT1.md5 | hashrat -lm -memcached 127.0.0.1
```

Read a list of hashes from stdin, and register them in a memcached server for later use in a search/check.

```
	cat APT1.ioc | hashrat -lm -memcached 127.0.0.1
```

Extract hashes from an Open IOC file and register them in a memcached server for later use in a search/check.

```
	hashrat -m -memcached 127.0.0.1 -r /
```

Search recursively for files whose hashes are stored in a memcached server.

```
	hashrat -devmode -whirlpool -64 /dev/sda1
```

                Generate a whirlpool hash of the entire device /dev/sda1. Output result in base 64.

```
        hashrat -sha1 -net ssh:user:password@myhost/bin/*
```

                Generate sha1 hashes of files in /bin/* on the remote machine 'myhost'

```
        hashrat -whirlpool -net http://myhost.com/webpage.html
```

                Generate whirlpool hash for the listed URL. (Note, many webpages have dynamic content that changes every time, so this will only return the same hash over and over if the page is static and doesn't change.

```
        hashrat -dups -r /home -u xattr
```

                Search for duplicate files under /home. Update hashes stored in filesystem attributes as you go

USES FOR HASHRAT

        1) Strong Passwords

        I mostly use hashrat to generate strong passwords for websites. As I always have access to hashrat, I don't have to remember the strong password, as I can always regenerate them with hashrat. So, I remember a handful of moderately decent passwords (i.e. things that I can't find by grepping in the '10,000 most popular passwords' list https://github.com/discourse/discourse/blob/master/lib/common_passwords/10k-common-passwords.txt), and I also remember a 'personal pin'. I then combine the website name, one of my passwords, and my personal pin, into a string and feed them into hashrat:

```
                echo "facebook.com password 1234" | hashrat -sha1 -64
```

        Obviously, my password isn't 'password' and my pin isn't '1234', but you get the idea. This gives me a 28-character string that should take "8.02 trillion trillion centuries" to crack with a "massive cracking array" according to Steve Gibson's 'Password haystacks' utility, https://www.grc.com/haystack.htm. This is what I then use as my password. Unfortunately some websites won't take a 28-character password, and for these I have to truncate to the appropriate length (using the -n flag), but the results are still stronger than anything I could remember, and nothing needs storing on disk (as with password managers).

        There are some dangers to using the 'echo' method shown above if you are on a shared machine, or if someone gets hold of your computer/harddrive. On a shared machine someone could type 'ps ax' to see all commands running, and if they time it right, they might see your command-line with your password in it. Another danger lies in using a shell (like bash) that will record your typed commands so you can recall them later. Bash stores this information on disk in the file .bash_history, so if you use the 'echo' method shown above your password will be saved on disk. To combat this hashrat has 'line mode'

```
                hashrat -sha1 -64 -lines
```

        This reads lines from stdin, so type into hashrat and then press 'enter', and you'll be given the hash of the line you typed. By this method your password is neither visible in 'ps ax', nor is ever stored on disk.

        '-lines' will produce a different hash to the 'echo' method listed above, because it strips any trailing whiespace off the lines read. If you want strict compatiblity with 'echo' (by default echo adds a 'newline' to the end of the text to output) then use 'rawlines' mode:

```
                hashrat -sha1 -64 -rawlines
```

Finally, you can prevent shoulder-surfers seeing you type your password by using the '-hide-input' or '-star-input' options to hide what you type.

2) Watching for file changes

Like md5sum/shasum etc, hashrat can be used to detect changes in files that might indicate malicious activity. For instance, in order to get early warning of malware like 'cryptolocker' (that encrypts files on a users disk, or on network shares, and then demands a ransom for file recovery) I scatter about the disk a number of 'Canary files' that should not change. I record their hashes and regularly check them. If they change, I know something is going on.

Hashes generated by hashrat can be output to a file, or stored in extended file attributes, or in a memcached server.

```
                hashrat -sha256 -r . > /tmp/files.sha256
                hashrat -sha256 -r . -xattr
                hashrat -sha256 -r . -memcached
```

Similarly these can then be used to check files later

```
                cat /tmp/files.sha256 | hashrat -c -sha256
                cat /tmp/files.sha256 | hashrat -sha256 -C /tmp
                hashrat -C -sha256 -xattr
                hashrat -C -sha256 -memcached
```

Hashrat has four 'check' options:

```
        -c   check from list
        -cf  check from list and only show failures
        -C   check directory
        -Cf  check directory and only show failures
```

the -c forms only check the files listed on stdin. The -C forms read a list of files on stdin, but they expect to be given a directory to check, and they check every file in that directory, outputing any that weren't in the list, or that are in the list but not on disk, as well as any that have changed. Thus the -C forms allow one to detect new files, changed files, and deleted files, where the -c form only detects changed and deleted files. The -c and -C forms both work as -C if the hashes are supplied via memcached or file system attributes (using -xattr or -memcached)

Note that -C implies -r, so you don't need to supply -r

3) Finding files that match hashes.

Using the -m flag hashrat can be told to read a range of hashes from stdin, and then search for files matching those hashes. For Example:

```
                cat APT1-AppendixE-MD5s.txt     | hashrat -r -m /usr
```

Will search recursively under /usr for files with hashes matching those in APT1-AppendixE-MD5s.txt. The input on stdin must begin with a hash, anything written after the hash will be treated as a comment to be displayed if a file matching the hash is found.

Hashtypes other than md5 can be used thusly:

```
                cat sha1-list.lst | hashrat -r -sha1 -m /usr
```

The input file can be in hashrat format, 'traditional' md5sum/shasum format, bsdsum format, or can be an Open IOC file from which any hashes will be extracted.

Hashes can also be loaded into a memcached server, so that the same file list can be checked on a number of machines, without needing to store the hashlist on those machines. First we load the hashes:

```
                cat APT1-AppendixE-MD5s.txt     | hashrat -lm -memcached
192.168.1.5
```

Loads the hashes to a memcached server at 192.168.1.5. We can then search against the memcached server by:

```
                hashrat -r -m -memcached 192.168.1.5 /usr
```

4) Find duplicate files

Using the -dups flag (usually in combination with the '-r' recursive flag) hashrat can be set to search for duplicate files and output any found to stdout.

5) CGI Mode

If hashrat is run with the -cgi flag, or if it's run with a name of 'hashrat.cgi' (either by renaming the 'hashrat' executable, or via a symbolic link) it will output a webpage that allows users to look up hashes over the web. This allows me to look-up my strong passwords even if I don't have access to a local version of hashrat.

6) As an 'ls'

Hashrat outputs a file's name, type, mode, mtime, uid, gid and size, along with a hash. This allows it to be used as a kind of 'ls' by ftp style programs, listing all the details of a file, but with the added feature of a hash.

7) Hashing files on remote machines

If run with the '-net' option, hashrat will treat paths starting with 'http://' or 'ssh://' differently, connecting to the target machine and pulling files off it (hashrat assumes there is no hashing program on the remote machine, and that it must therefore download the files to hash them). For ssh paths wildcars are supported:

```
                hashrat -net ssh://username:password@server/usr/bin/*
```

EXTENDED FILESYSTEM ATTRIBUTES (XATTR)

Hashrat can use extended filesystem attributes where these are supported. This allows a hash to be stored in the filesystem metadata of the target file. This can then be used for checking hashes, or for caching hashes to produce faster output during hashing runs. There are two types of filesystem attribute, 'trusted' attributes, which can only be set and read by root, and 'user' attributes, which can be set and read by any user that has the appropriate permissions for the file.

Hashes can be stored against files by using the -xattr option to set 'user' attributes...

```
                hashrat -sha256 -r . -xattr
```

... and using the '-txattr' flag to set trusted attributes (you must be root to set trusted attributes)

```
                hashrat -sha256 -r . -txattr
```

When checking either flag can be used, but hashrat will always use 'trusted' attributes when running as root, if those are available, otherwise it will fall back to 'user' attributes.

```
                hashrat -c -sha256 -r . -xattr
```

The -cache option allows using stored hashes rather than regenerating hashes. It only considers hashes stored in 'user' attributes at current.

```
                hashrat -r . -cache
```

This makes getting a report of hashes considerably faster, but it runs the risk that the hashes may not be accurate. Hashrat will only output a hash stored in file attributes if the storage time of the hash is younger than the modify time (mtime) of the file, however, this means an attacker could change the modify time of the file to hide changes they've made. Thus this feature should not be used for security checking purposes (but should be safe for uses like finding files that have changed and need to be backed up, for instance).

The '-u' option allows filesystem attributes to be updated as we do checks (in check mode -xattr means 'read from xattr', so we need the '-u' (update) flag to tell hashrat to also update the stored hash of any files who's hash has changed).

```
        hashrat -c -r . -xattr -u xattr
```