The `find` command is one of the most powerful and versatile tools in Linux for searching for files and directories based on various criteria. It's often used in scripts for automation and for system administration tasks like cleaning up old files or finding large files.

The basic syntax for the `find` command is:

Bash

```
find [starting_point...] [expression]
```

- `starting_point` : This is the directory (or directories) where `find` will begin its search. It will recursively search all subdirectories within this path. A common starting point is `.` (current directory) or `/` (root directory, for a system-wide search).
- `expression` : This is where you define the criteria for your search (e.g., name, type, size, modification time, permissions) and actions to perform on the found files.

Let's break down some common parameters (options and tests) with examples:

Basic Search Parameters

1. `-name "pattern"` : **Search by Name (Case-Sensitive)**
   Finds files or directories with an exact name match. Wildcards (`*` , `?`) can be used.
   - **Example 1: Find a file named `my_document.txt` in the current directory and its subdirectories.**

     Bash

     ```
     find . -name "my_document.txt"
     ```

   - **Example 2: Find all `.log` files in the `/var/log` directory.**

     Bash

     ```
     find /var/log -name "*.log"
     ```

2. `-iname "pattern"` : **Search by Name (Case-Insensitive)**
   Similar to `-name` , but ignores case.
   - **Example: Find all files named `report.pdf` , `Report.pdf` , `REPORT.PDF` , etc., in your home directory.**

     Bash

```
find ~ -iname "report.pdf"
```

3. `-type [f|d|l|c|b|s|p]` : **Search by File Type**
   Filters results by specific file types:
   - `f` : regular file
   - `d` : directory
   - `l` : symbolic link
   - `c` : character device
   - `b` : block device
   - `s` : socket
   - `p` : named pipe
   - **Example 1: Find all directories in the current directory.**

     Bash

     ```
     find . -type d
     ```

   - **Example 2: Find all symbolic links in `/usr/local/bin` .**

     Bash

     ```
     find /usr/local/bin -type l
     ```

Search by Size

The `-size` option allows you to search for files based on their size. Units can be specified:

- `c` : bytes
- `k` : kilobytes
- `M` : megabytes
- `G` : gigabytes

You can use `+` before the number for "greater than" and `-` for "less than".

- **Example 1: Find all files larger than 100MB in your home directory.**

  Bash

```
find ~ -type f -size +100M
```

- **Example 2: Find all files exactly 5KB in size in the current directory.**

  Bash

  ```
  find . -type f -size 5k
  ```

- **Example 3: Find files between 1GB and 5GB in the `/backup` directory.**

  Bash

  ```
  find /backup -type f -size +1G -size -5G
  ```

Search by Time

These options relate to the timestamps associated with files:

- `-mtime n` : Files modified `n` days ago.
- `-atime n` : Files last accessed `n` days ago.
- `-ctime n` : Files whose status (permissions, ownership, etc.) was last changed `n` days ago.
- `-mmin n` : Files modified `n` minutes ago.
- `-amin n` : Files last accessed `n` minutes ago.
- `-cmin n` : Files whose status was last changed `n` minutes ago.

  Use `+n` for "more than n days/minutes ago" and `-n` for "less than n days/minutes ago" (i.e., within the last n days/minutes).

- **Example 1: Find files modified within the last 7 days in the current directory.**

  Bash

  ```
  find . -type f -mtime -7
  ```

- **Example 2: Find files accessed more than 30 days ago in `/tmp` .**

  Bash

  ```
  find /tmp -atime +30
  ```

- **Example 3: Find files changed exactly 2 days ago.**

  Bash

  ```
  find . -ctime 2
  ```

- **Example 4: Find files modified in the last 60 minutes.**

  Bash

  ```
  find . -mmin -60
  ```

Search by Permissions

`find` can search for files with specific permission modes.

- `-perm mode` : Finds files with permissions exactly matching `mode` .
- `-perm -mode` : Finds files where *all* the bits in `mode` are set (i.e., the permissions are at least `mode`).
- `-perm /mode` : Finds files where *any* of the bits in `mode` are set.
- **Example 1: Find files with exact permissions 644 (rw-r--r--) in your home directory.**

  Bash

  ```
  find ~ -type f -perm 644
  ```

- **Example 2: Find directories that are writable by group and others (at least 777, 775, etc.).**

  Bash

  ```
  find . -type d -perm -002 # or -perm -g+w,o+w
  ```

- **Example 3: Find files that are executable by anyone (permission bit `1` in owner, group, or others).**

  Bash

  ```
  find . -type f -perm /111
  ```

Search by Ownership

- `-user username` : Finds files owned by a specific user.
- `-group groupname` : Finds files owned by a specific group.
- **Example 1: Find all files owned by the user `john` in `/home`.**

  Bash

  ```
  find /home -user john
  ```

- **Example 2: Find all files in `/var/www` owned by the `apache` group.**

  Bash

  ```
  find /var/www -group apache
  ```

Advanced Options and Actions

1. `-empty` : Finds empty files or directories.
   - **Example 1: Find all empty files in the current directory.**

     Bash

     ```
     find . -type f -empty
     ```

   - **Example 2: Find and delete all empty directories in `/tmp`.**

     Bash

     ```
     find /tmp -type d -empty -delete
     ```

2. `-delete` : Deletes the found files/directories. **Use with extreme caution!** It does not ask for confirmation.
   - **Example: Delete all `.tmp` files older than 30 days.**

     Bash

```
find . -name "*.tmp" -mtime +30 -delete
```

3. `-exec command {} \;` : Executes a command on each found file.
   - `{}` : A placeholder that `find` replaces with the current file path.
   - `\;` : Terminates the `-exec` command.
   - **Example 1: Change permissions of all `.sh` files to 755.**

     Bash

     ```
     find . -name "*.sh" -exec chmod 755 {} \;
     ```

   - **Example 2: Copy all `.jpg` files found to a new directory `/backup/images` .**

     Bash

     ```
     find . -name "*.jpg" -exec cp {} /backup/images \;
     ```

4. `-exec command {} +` : Executes a command on a batch of found files (more efficient than `\;`).
   - This is generally preferred over `\;` when the executed command can handle multiple arguments (like `rm` , `chmod` , `cp`).
   - **Example: Delete all `.bak` files found (more efficiently).**

     Bash

     ```
     find . -name "*.bak" -exec rm {} +
     ```

5. `-print` : Prints the full path of the found files. (This is the default action if no other action like `-exec` or `-delete` is specified).
   - **Example: Explicitly print all files named `important.txt` .**

     Bash

     ```
     find / -name "important.txt" -print 2>/dev/null
     ```

     (The `2>/dev/null` redirects permission denied errors to `/dev/null` so they don't clutter the output.)
6. `-prune` : Prevents `find` from descending into a specific directory. Useful for excluding certain paths from the search.

- **Example: Find all `.txt` files in the current directory, but exclude the `old_data` subdirectory.**

```bash
find . -path "./old_data" -prune -o -name "*.txt" -print
```

(Here, `-o` acts as an "OR" operator. It says: if the path is `old_data`, prune it; otherwise, if the name matches `*.txt`, print it.)

7. `-mindepth N` **and** `-maxdepth N`: Controls the depth of the search.
   - `N` is an integer representing the depth level.
   - `-mindepth 1` means start searching from the current directory (level 1), not the current directory itself (level 0).
   - `-maxdepth 1` means search only the current directory, not its subdirectories.
   - **Example 1: Find files only in the current directory (not subdirectories).**

```bash
find . -maxdepth 1 -type f
```

   - **Example 2: Find directories that are at least 2 levels deep but no more than 3 levels deep from the starting point.**

```bash
find . -mindepth 2 -maxdepth 3 -type d
```

Combining Conditions (Logical Operators)

You can combine multiple criteria using logical operators:

- `-a` **(or omit, it's the default)**: AND (both conditions must be true)
- `-o`: OR (at least one condition must be true)
- `!`: NOT (negates the following condition)
- `\( expression \)`: Groups expressions for precedence. Parentheses must be escaped (`\(` and `\)`) because they have special meaning in the shell.
- **Example 1: Find all `.txt` files that are larger than 1MB.**

Bash

```
find . -name "*.txt" -size +1M
# (or explicitly: find . -name "*.txt" -a -size +1M)
```

- **Example 2: Find all `.jpg` or `.png` files.**

  Bash

  ```
  find . -type f \( -name "*.jpg" -o -name "*.png" \)
  ```

- **Example 3: Find all files that are NOT hidden (i.e., don't start with a dot).**

  Bash

  ```
  find . -type f ! -name ".*"
  ```

- **Example 4: Find all `.conf` files modified in the last 3 days, but not owned by `root`.**

  Bash

  ```
  find /etc -name "*.conf" -mtime -3 ! -user root
  ```

The `find` command is incredibly versatile, and mastering it will significantly boost your productivity in the Linux command line. Always be careful when using options like `-delete` or `-exec rm`, as they can lead to irreversible data loss. It's often a good practice to run `find` without the deletion/modification part first to ensure it's matching the correct files.