find . -name "\*.py" -exec perl -ne 'print "$ARGV:$.:$\_" if /regex-C-xlwings\.dll/' {} +

Can you convert above command to use grep (with pcre 2) instead of perl Will it be fully flexible like using perl with pcre2?

Edit

Here's the equivalent command using grep with PCRE2:

bash

find . -name "\*.py" -exec grep -Hn --perl-regexp 'regex-C-xlwings\.dll' {} +

Or with the short options:

bash

find . -name "\*.py" -exec grep -HnP 'regex-C-xlwings\.dll' {} +

**Explanation of the conversion:**

* -H = Always print filename (equivalent to $ARGV in Perl)
* -n = Print line numbers (equivalent to $. in Perl)
* -P or --perl-regexp = Use Perl-compatible regular expressions (PCRE)
* The {} and + remain the same for find -exec

**Regarding flexibility compared to Perl with PCRE2:**

Grep with PCRE2 is **almost as flexible** as Perl with PCRE2 for pattern matching, but there are some differences:

**What's equivalent:**

* Basic regex syntax and most PCRE features
* Lookaheads, lookbehinds, atomic groups
* Unicode support
* Most character classes and modifiers

**Where Perl has advantages:**

* More advanced text processing capabilities beyond just matching
* Complex substitutions and transformations
* Built-in variables and automatic field splitting
* More sophisticated conditional logic within the regex engine
* Some newer PCRE2 features might be available in Perl before grep

**Where grep might have advantages:**

* Often faster for simple pattern matching
* Lower memory overhead
* Simpler syntax for basic use cases

For your specific use case (finding a literal string in files), grep with PCRE2 will be fully adequate and likely perform better than the Perl approach.

Retry

Claude does not have the ability to run the code it generates yet.

D

Give examples of these where we cannot do it with grep + pcre2:

**Where Perl has advantages:** \* More advanced text processing capabilities beyond just matching \* Complex substitutions and transformations \* Built-in variables and automatic field splitting \* More sophisticated conditional logic within the regex engine \* Some newer PCRE2 features might be available in Perl before grep

Edit

Here are concrete examples where Perl's capabilities exceed what grep + PCRE2 can do:

**1. More advanced text processing capabilities beyond just matching**

**Perl example:**

perl

find . -name "\*.py" -exec perl -ne '

chomp;

my @words = split;

if (/regex-C-xlwings\.dll/) {

print "$ARGV:$.:$\_ (found in context: prev=" . ($words[$.-2] // "N/A") . ")\n";

}

' {} +

**What it does:** Finds matches but also analyzes context, word positions, etc. **Grep limitation:** Can only match and print lines, cannot perform complex analysis of the matched content.

**2. Complex substitutions and transformations**

**Perl example:**

perl

find . -name "\*.py" -exec perl -pe '

s/(regex-C-xlwings\.dll)/uc($1)/ge if /some\_condition/;

s/(\w+)=(\d+)/sprintf("%s=%04d", $1, $2)/ge;

' {} +

**What it does:** Performs conditional substitutions with function calls, formatting, etc. **Grep limitation:** Cannot modify content - grep is read-only for searching.

**3. Built-in variables and automatic field splitting**

**Perl example:**

perl

find . -name "\*.py" -exec perl -ane '

if (/regex-C-xlwings\.dll/) {

print "$ARGV:$.:Field count: " . scalar(@F) .

", Last field: $F[-1], Total chars: " . length($\_) . "\n";

}

' {} +

**What it does:** Uses @F (auto-split fields), $. (line number), length(), etc. **Grep limitation:** No automatic field splitting or rich built-in variables beyond basic match info.

**4. More sophisticated conditional logic within the regex engine**

**Perl example:**

perl

find . -name "\*.py" -exec perl -ne '

if (/(?{$count++})regex-C-xlwings\.dll(?(?{$count > 5})SUCCESS|FAIL)/) {

print "$ARGV:$.:Found on attempt $count\n";

}

BEGIN { $count = 0 }

' {} +

**What it does:** Uses embedded code (?{...}) and conditional regex (?(?{condition})yes|no). **Grep limitation:** PCRE2 has some code execution features, but not as extensive as Perl's.

**5. Stateful processing across lines**

**Perl example:**

perl

find . -name "\*.py" -exec perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' {} +

**What it does:** Maintains state across files, provides summary statistics. **Grep limitation:** Each grep invocation is independent - cannot maintain state across files or provide aggregated reports.

**6. Complex data structures and processing**

**Perl example:**

perl

find . -name "\*.py" -exec perl -ne '

BEGIN { %dll\_context = (); }

if (/^(\s\*)(.\*regex-C-xlwings\.dll.\*)/) {

my $indent = length($1);

push @{$dll\_context{$ARGV}}, {

line => $.,

content => $2,

indent\_level => $indent

};

}

END {

for my $file (keys %dll\_context) {

print "File: $file\n";

for my $match (@{$dll\_context{$file}}) {

print " Line $match->{line} (indent: $match->{indent\_level}): $match->{content}\n";

}

}

}

' {} +

**What it does:** Builds complex data structures, analyzes indentation patterns, provides structured output. **Grep limitation:** Cannot build data structures or perform complex multi-pass analysis.

**Real-world scenario where you'd need Perl:**

If you wanted to find all DLL references and generate a report showing:

* Which files contain them
* How many times each DLL appears
* The context (function calls) around each reference
* Whether they're in comments or actual code
* A summary sorted by frequency

This kind of analysis requires Perl's full programming capabilities, which grep + PCRE2 simply cannot provide.

**Grep's strength:** Fast, simple pattern matching and filtering. **Perl's strength:** Full programming language for complex text processing and analysis.

Retry

Claude does not have the ability to run the code it generates yet.

D

Are above examples recursive or they work on standalone locations?

Edit

The examples I provided are **recursive** because they all use:

bash

find . -name "\*.py" -exec perl ... {} +

The find . command:

* Starts from current directory (.)
* Recursively searches all subdirectories
* Finds all files matching \*.py pattern
* Executes the Perl command on each found file

**To make them work on standalone locations:**

**Option 1: Specific files**

bash

perl -ne 'your\_code\_here' file1.py file2.py file3.py

**Option 2: Single directory (non-recursive)**

bash

perl -ne 'your\_code\_here' \*.py

**Option 3: Multiple specific directories**

bash

perl -ne 'your\_code\_here' /path/to/dir1/\*.py /path/to/dir2/\*.py

**Option 4: Custom find with specific paths**

bash

find /specific/path1 /specific/path2 -maxdepth 1 -name "\*.py" -exec perl -ne 'your\_code\_here' {} +

(The -maxdepth 1 prevents recursion)

**Example conversion:**

**Original (recursive):**

perl

find . -name "\*.py" -exec perl -ne '

if (/regex-C-xlwings\.dll/) {

print "$ARGV:$.:$\_";

}

' {} +

**Standalone version:**

perl

perl -ne '

if (/regex-C-xlwings\.dll/) {

print "$ARGV:$.:$\_";

}

' specific\_file.py another\_file.py

The Perl code itself doesn't change - only how you invoke it and which files you pass to it. The find command is what makes it recursive, not the Perl script.

Retry

Claude does not have the ability to run the code it generates yet.

D

I tried this option but it does not show in which folder the file is found:

juhi@LAPTOP-BGI3SJP4 MSYS ~ $ find . -name "\*.py" -exec perl -ne ' BEGIN { %seen\_files = (); $total\_matches = 0; } if (/regex-C-xlwings\.dll/) { $seen\_files{$ARGV}++; $total*matches++; print "$ARGV:$.:$*"; } END { print "Summary: $total\_matches matches across " . scalar(keys %seen\_files) . " files\n"; } ' {} + Summary: 0 matches across 0 files ./Downloads/regex-test.py:15168:dll = ffi.dlopen("D:\Programs\Msys2\home\juhi123\Downloads\regex-C-xlwings.dll") Summary: 1 matches across 1 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files Summary: 0 matches across 0 files

Edit

The issue you're experiencing is caused by how find ... -exec ... {} + works. The + at the end batches multiple files together, but Perl's BEGIN and END blocks execute once per **batch**, not once per entire operation.

Here are several solutions:

**Solution 1: Use \; instead of + (Simple fix)**

bash

find . -name "\*.py" -exec perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' {} \;

**Downside:** This runs Perl once per file, so you'll get multiple summaries.

**Solution 2: Remove BEGIN/END blocks (Recommended)**

bash

find . -name "\*.py" -exec perl -ne '

if (/regex-C-xlwings\.dll/) {

print "$ARGV:$.:$\_";

}

' {} +

Then use shell tools for summary:

bash

find . -name "\*.py" -exec perl -ne '

if (/regex-C-xlwings\.dll/) {

print "$ARGV:$.:$\_";

}

' {} + | tee /tmp/results.txt && echo "Summary: $(wc -l < /tmp/results.txt) matches across $(cut -d: -f1 /tmp/results.txt | sort -u | wc -l) files"

**Solution 3: Two-step approach (Best for complex analysis)**

**Step 1:** Find all matching files and save to temp file:

bash

find . -name "\*.py" > /tmp/pyfiles.txt

**Step 2:** Process all files in one Perl invocation:

bash

perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' $(cat /tmp/pyfiles.txt)

**Solution 4: Use xargs (Alternative)**

bash

find . -name "\*.py" -print0 | xargs -0 perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

'

**Why your current output shows the path**

The path **is** showing correctly: ./Downloads/regex-test.py:15168:...

The ./Downloads/ part **is** the folder path. The issue is just the multiple summary lines due to batching.

**Recommendation:** Use Solution 2 or 3 depending on whether you need the summary statistics.

Retry

Claude does not have the ability to run the code it generates yet.

D

Can you not combine Step 1 and Step 2 (integrate) into a single command?

Edit

Yes, absolutely! Here are several ways to combine them into a single command:

**Solution 1: Using command substitution (Most straightforward)**

bash

perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' $(find . -name "\*.py")

**Solution 2: Using xargs with null delimiter (Handles filenames with spaces)**

bash

find . -name "\*.py" -print0 | xargs -0 perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

'

**Solution 3: Using process substitution (bash)**

bash

perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' < <(find . -name "\*.py" -exec cat {} +)

**Solution 4: Let Perl do the file finding (Perl-only solution)**

perl

perl -MFile::Find -e '

my %seen\_files = ();

my $total\_matches = 0;

find(sub {

return unless /\.py$/;

open my $fh, "<", $\_ or return;

my $line\_num = 0;

while (my $line = <$fh>) {

$line\_num++;

if ($line =~ /regex-C-xlwings\.dll/) {

$seen\_files{$File::Find::name}++;

$total\_matches++;

print "$File::Find::name:$line\_num:$line";

}

}

close $fh;

}, ".");

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

'

**Recommendation**

**For MSYS2/Windows:** Use **Solution 1** (command substitution) - it's the simplest and most portable:

bash

perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' $(find . -name "\*.py")

This will give you exactly what you want:

* Single summary at the end
* Full paths showing folders
* All in one command
* Works reliably in MSYS2

**Note:** If you have filenames with spaces, use Solution 2 with xargs instead.

Retry

Claude does not have the ability to run the code it generates yet.

D

But whether they use pcre2?

Edit

**No, none of those solutions use PCRE2.** They all use Perl's built-in regex engine, which is based on PCRE but is Perl's own implementation.

Here's the distinction:

**What you get with those Perl solutions:**

* **Perl's regex engine** (evolved from PCRE but now independent)
* All advanced Perl regex features
* Programming capabilities (variables, data structures, etc.)

**To actually use PCRE2, you have these options:**

**Option 1: grep with PCRE2 + shell processing**

bash

*# Find matches with grep + PCRE2*

results=$(find . -name "\*.py" -exec grep -HnP 'regex-C-xlwings\.dll' {} +)

echo "$results"

echo "Summary: $(echo "$results" | wc -l) matches across $(echo "$results" | cut -d: -f1 | sort -u | wc -l) files"

**Option 2: ripgrep (uses PCRE2 when available)**

bash

rg --pcre2 'regex-C-xlwings\.dll' --glob '\*.py' --line-number --with-filename

**Option 3: pcregrep (pure PCRE2 implementation)**

bash

find . -name "\*.py" -exec pcregrep -Hn 'regex-C-xlwings\.dll' {} +

**The key question is: Do you specifically need PCRE2?**

**If you need PCRE2 specifically:**

* Use grep with -P, ripgrep with --pcre2, or pcregrep
* You'll lose the advanced programming capabilities (summary stats, complex processing)

**If you just need powerful regex matching:**

* Perl's regex engine is actually **more advanced** than PCRE2
* You get the programming capabilities as a bonus
* Perl's regex engine supports features that PCRE2 doesn't have

**For your original use case:**

bash

*# This uses PCRE2 (simple matching only)*

find . -name "\*.py" -exec grep -HnP 'regex-C-xlwings\.dll' {} +

*# This uses Perl's engine (more features + programming)*

perl -ne '...' $(find . -name "\*.py")

**Bottom line:** If you specifically need PCRE2 compliance, use grep/ripgrep/pcregrep. If you want the most powerful solution, Perl's regex engine is actually superior to PCRE2.

Retry

Claude does not have the ability to run the code it generates yet.

D

This one worked as your recommend because I am on msys on windows - But whether this will also work in any Linux distro?

juhi@LAPTOP-BGI3SJP4 MSYS ~ $ perl -ne ' BEGIN { %seen\_files = (); $total\_matches = 0; } if (/regex-C-xlwings\.dll/) { $seen\_files{$ARGV}++; $total*matches++; print "$ARGV:$.:$*"; } END { print "Summary: $total\_matches matches across " . scalar(keys %seen\_files) . " files\n"; } ' $(find . -name "\*.py") ./Downloads/regex-test.py:248796:dll = ffi.dlopen("D:\Programs\Msys2\home\juhi123\Downloads\regex-C-xlwings.dll") ./xlwings-dll.py:7053768:lib\_path = "regex-C-xlwings.dll" # Adjust the path to your DLL Summary: 2 matches across 2 files

Edit

**Yes, this command will work on virtually all Linux distributions** because it uses only standard, portable components:

**What makes it portable:**

1. **perl** - Available on virtually every Linux distribution by default
2. **find** - Part of GNU findutils, standard on all Linux systems
3. **Command substitution $()** - Standard POSIX shell feature
4. **Basic Perl regex** - Uses Perl's built-in regex engine (not external dependencies)

**Linux distributions where this will work:**

* Ubuntu/Debian (all versions)
* RHEL/CentOS/Rocky/AlmaLinux
* Fedora
* SUSE/openSUSE
* Arch Linux
* Alpine Linux
* Gentoo
* And virtually any other Linux distribution

**Only potential issues (very rare):**

**1. Minimal containers/systems:**

Some ultra-minimal Docker images might not have Perl installed:

bash

*# Check if perl is available*

which perl

*# or*

perl --version

**2. Very old systems:**

Systems from 15+ years ago might have older Perl versions, but the regex features you're using are ancient and well-supported.

**3. Different shell behavior:**

If someone is using a very old shell (not bash/dash/zsh), command substitution might behave slightly differently, but this is extremely rare.

**To make it even more portable:**

bash

*# Add error checking*

if command -v perl >/dev/null 2>&1; then

perl -ne '

BEGIN { %seen\_files = (); $total\_matches = 0; }

if (/regex-C-xlwings\.dll/) {

$seen\_files{$ARGV}++;

$total\_matches++;

print "$ARGV:$.:$\_";

}

END {

print "Summary: $total\_matches matches across " .

scalar(keys %seen\_files) . " files\n";

}

' $(find . -name "\*.py")

else

echo "Perl not found. Please install perl."

fi

**Alternative if Perl is missing:**

On the rare system without Perl, you could fall back to:

bash

*# Using only grep (PCRE2) + shell*

matches=$(find . -name "\*.py" -exec grep -HnP 'regex-C-xlwings\.dll' {} + 2>/dev/null)

echo "$matches"

if [ -n "$matches" ]; then

echo "Summary: $(echo "$matches" | wc -l) matches across $(echo "$matches" | cut -d: -f1 | sort -u | wc -l) files"

else

echo "Summary: 0 matches across 0 files"

fi