

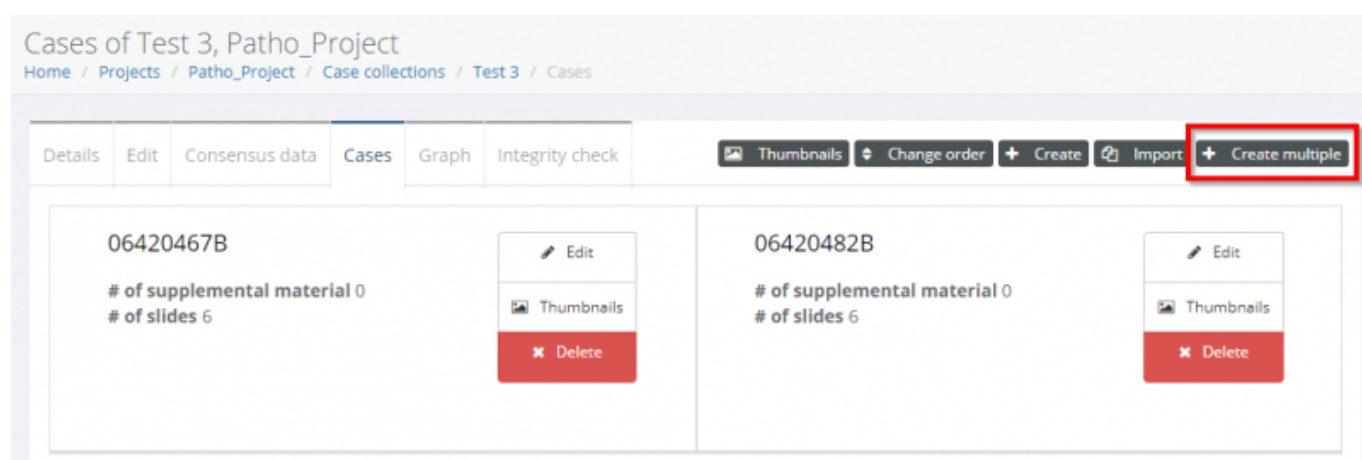
## Creating (Multiple) Cases with Pattern Search

Sometimes you may need to create many Cases from WSIs that aren't in their own subfolder, or create a Case that has slides across several different folders, or find a large number of examples of a particular stain, indication, or filetype.

That's where Pattern Search can make your life a lot easier by eliminating the manual searching through all your directories to add them one by one.

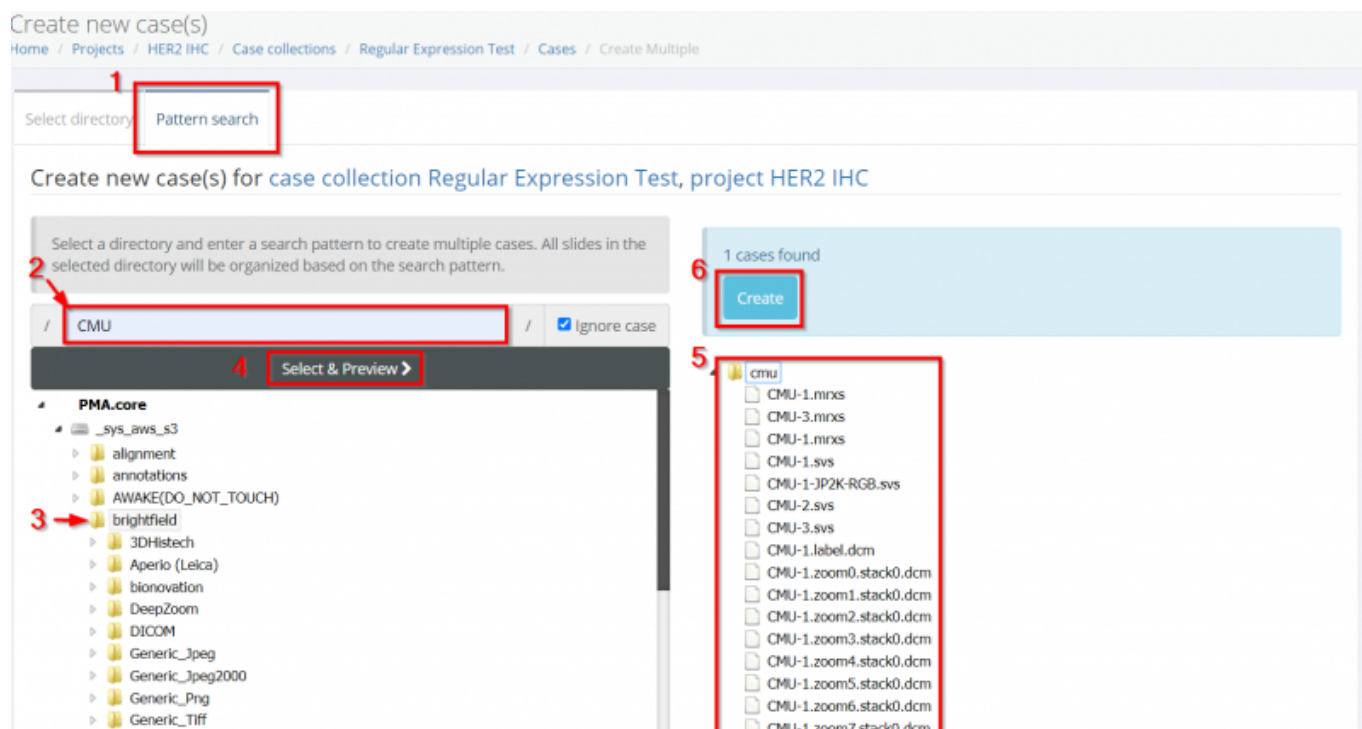
If you're not sure how to get to the Cases menu pictured below, first read [Create Cases and Case Collections](#), then come back here.

Then click the "Create Multiple" button.



### Running a pattern search

1. First click the 'Pattern Search' tab at the top of the page
2. Type in your search term, this is the text you will be searching for. The pattern search follows 'regular expression' rules. More information on how to run these is at the bottom of this page
3. Third select the directory you want to run your search in. The pattern search will be applied to every file in every (sub)folder
4. Click 'Select and Preview' to run the search
5. Expand the folder view to see the files your search has yielded
6. When you're happy with the results, click Create. If you want to add/remove files you can edit the case after creation.



## Pattern Search Behaviour and how to use 'Regular Expression' in a Search

Regular expressions, often abbreviated as regex or regexp, are powerful tools for pattern matching and text manipulation. They provide a concise and flexible way to describe, search, and manipulate strings.

Some examples include:

Using \$ at the end of your search term to match the end of the filename, e.g. .filetype\_name\$ will only include files ending in .filetype\_name

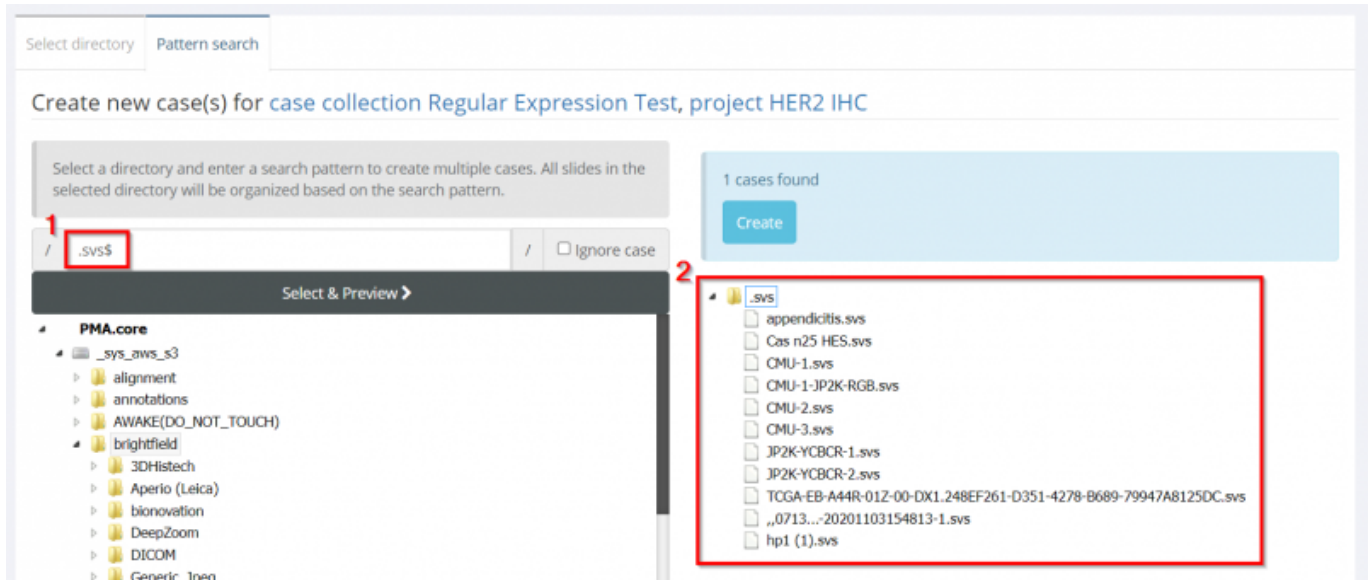
Using ^CF to match any string starting with 'CF'

Using ? to indicate subexpressions in your pattern, e.g. "fish(er)?" matches "fish" or "fisher".

You can learn more about regex with online guides including:

<https://www.microfocus.com/documentation/relativity/relativity1216/reldbdsn/GUID-7C2DF185-41A1-4448-81E7-3252AA8DEBB3.html>

In the screenshot below you can see the \$ regex [1] has been used to select only .svs files across many subfolders [2]



## Regular Expression Syntax

Here are some key concepts and syntax:

### 1. Literal Characters:

Most characters in a regular expression match themselves literally.

For example, the regular expression `abc` will match the string "abc" in the input.

### 2. Metacharacters:

- Certain characters have special meanings in regular expressions and are called metacharacters. Some common ones include:
  - `.` (dot): Matches any single character except a newline.
  - `*`: Matches 0 or more occurrences of the preceding character or group.
  - `+`: Matches 1 or more occurrences of the preceding character or group.
  - `?`: Matches 0 or 1 occurrence of the preceding character or group.
  - `^`: Anchors the regex at the beginning of the line.
  - `$`: Anchors the regex at the end of the line.

### 3. Character Classes:

- `[ ]`: Defines a character class. For example, `[aeiou]` matches any vowel.
- `[^ ]`: Negates a character class. For example, `[^0-9]` matches any non-digit character.

### 4. Quantifiers:

- `{n}`: Matches exactly `n` occurrences of the preceding character or group.
- `{n,}`: Matches `n` or more occurrences.
- `{n,m}`: Matches between `n` and `m` occurrences.

### 5. Grouping and Capturing:

- `( )`: Groups expressions together. Also used for capturing portions of the

matched text.

## 6. Anchors:

- `\b`: Word boundary anchor.
- `\B`: Non-word boundary anchor.
- `^` and `$`: Anchors for the start and end of a line, respectively.

## 7. Escape Characters:

- `\`: Escapes a metacharacter, allowing it to be treated as a literal character. For example, `\.` matches a literal dot.

## 8. Modifiers:

- `i`: Case-insensitive matching.
- `g`: Global matching (find all matches rather than stopping after the first match).

## 9. Special Sequences:

- `\d`: Matches any digit (equivalent to `[0-9]`).
- `\w`: Matches any word character (alphanumeric + underscore).
- `\s`: Matches any whitespace character.

## 10. Lookaheads and Lookbehinds:

- `(?=...)`: Positive lookahead assertion.
- `(?!...)`: Negative lookahead assertion.
- `(?<=...)`: Positive lookbehind assertion.
- `(?<!...)`: Negative lookbehind assertion.

Regular expressions are supported in many programming languages (e.g., Python, JavaScript, Java) and text editors. They are incredibly versatile but can also become complex. Practice and experimentation are key to mastering regex. Online regex testers, such as [regex101](https://regex101.com) or [RegExr](https://regexr.com), can help you test and visualize your regular expressions.

From:  
<https://docs.pathomation.com/pathotrainer/> - **Pathotrainer**

Permanent link:  
[https://docs.pathomation.com/pathotrainer/doku.php?id=creating\\_cases\\_using\\_pattern\\_search&rev=1709213463](https://docs.pathomation.com/pathotrainer/doku.php?id=creating_cases_using_pattern_search&rev=1709213463)

Last update: **2024/02/29 16:31**

