ECOL 553L

Advanced REGEX

Special Characters in Regular Expressions

.	Match any single character
^	Anchor match at beginning of string
$	Anchor match at end of string
?	Match preceding element 0 or 1 time
*	Match preceding element 0 or more times
+	Match preceding element 1 or more times
{n,m}	Match preceding element n to m times
[ ]	Match any character in character class
[^]	Match any character NOT in character class
( )	Group and capture expression
|	Match either expression preceding or following
\	Escape the character immediately following \

Review of Pattern Capturing

- Segments of a pattern surrounded by parentheses `( )` are captured in special temporary variables named `$1`, `$2`, `$3`, etc.
- We can match and capture repeated motifs and use the length function to compute the number of motifs found:

```
if ($seq =~ /(($motif){$min,})/) {
    print "Matched $motif at least $min times\n";
    my $num_matched = length($1)/length($motif);
    print "Found $num_matched $motif\n";
}
```

- When writing complex patterns, you can work from left to right or right to left, adding one pattern element at a time.
- Do not include extraneous white space in your patterns.
- If you are using `{min,max}` to quantify a part of the pattern that is a variable, parenthesize the variable to avoid confusion with hash syntax (as in the above example).

Further Review of Pattern Capturing

- What will be output by the following code? Notice that the pattern contains spaces before \d and before `[A-Z]`:

```
$data = "NG_011606  2126bp  DNA linear PRI 01-NOV-2009";
if ($data =~ /([A-Z]+).+( \d+)bp.+( [A-Z]+)/) {
    print "Found $1 $2 $3\n";
} else {
    print "Expected pattern not found.\n";
}
```

- Another way to write the pattern is:

```
if ($data =~ /([A-Z]+).+(\s\d+)bp\s.+(\s[A-Z]+)/)
```

- How can we modify the pattern so that it will also capture the date in this example?

Another Pattern Capturing Example

- Suppose that we want to count and classify files. Assuming that the files are named with extensions that reflect their types, we can use this code:

```
my $dir = $ARGV[0];
if (!defined $dir) { $dir = "." }
if (!-d $dir) { die "Usage: $0 dirname\n"; }
my @files = glob("$dir/*");
my %ftype;  # hash with file ext as key, count as value

foreach my $f (@files) {
    if ($f =~ /\.([^\.]*)$/) {  # match .ext at end of
    filename
        print "File: $f\tmatched ext: $1\n";
        $ftype{$1}++;
    } else {
        print STDERR "File $f: No match to pattern\n";
    }
}
```

Perl Pattern Substitution

- In addition to pattern matching capabilities, Perl can do pattern substitution. For substitution, you use `s` in front of the pattern, and provide the substitution string after the pattern, followed by a final `/`
- Only the first match to the pattern gets substituted unless the g modifier is specified.
- pattern substitution `($str =~ s/pattern/substitution/)`:
  - `$seq =~ s/CAG/1234/;`
    `# change 1st upper case CAG to 1234`
  - `$seq =~ s/CAG/1234/i;`
    `# change 1st mixed case CAG to 1234`
  - `$seq =~ s/CAG/1234/g;`
    `# change all occurrences of upper case CAG to 1234`
  - `$seq =~ s/CAG/1234/gi;`
    `# change all occurrences of mixed case CAG to 1234`
- Using pattern substitution to count occurrences
- Perl pattern substitution also counts the number of substitutions it finds:
  - `$count = ( $seq =~ s/ABC/VWXYZ/ );`
  - `$count = ( $seq =~ s/ABC/VWXYZ/g );`
  - `$c_count = ( $seq =~ s/C/C/gi );`
  - `$g_count = ( $seq =~ s/G/G/gi );`

Pattern Capture and Substitution Example

- You can use pattern capture to slice, dice and rearrange data. For example, suppose we have a FASTA file of sequences with identifiers that look like this:

```
>gi|8923664|ref|NM_017949.1| Homo sapiens CUE domain
```

- We want to output the Accession number without the version, followed by the GI number. Using pattern matching/capture, we could do so with the following code:

```
open(SFIL, $file) or die "Cannot open $file\n";
while ($line = <SFIL>) {
    # Capture the GI and Accession/version
    if ($line =~ /^>gi\|(\d+)\|[^\|]+\|([^\|]+/)) {
        $gi = $1; $acc = $2;
        # Substitute .version number with nothing!
        $acc =~ s/\.\d+//;
        print "Accession: $acc  GI: $gi\n";
    }
} # end while <SFIL>
```

Transliteration of characters

- Besides pattern matching and substitution, Perl has an easy way to transliterate characters in strings. For example, if you wanted to change a telephone number that uses letters into the numeric equivalent, you could use the code:

```
print "Enter word: ";
$word = <STDIN>;
chomp($word);
$tel_num = $word;
$tel_num =~ tr/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
                              22233344455566677778889999/;
print "Numeric Telephone number is: $tel_num \n";
```

- Transliteration makes it easy to complement a DNA sequence:

```
$seq = "ATGCCGCAGCAGTCAAGTCGTAGTG";
$seq =~ tr/ACGTacgt/TGCAtgca/;
```

- Note that with `tr` you don't need the `/g` modifier

More about Transliteration

- Like pattern substitution, `tr` with the binding operator returns the number of characters matched. To count the number of vowels in a string, you could use:

```
my $num_vowels = ($str =~ tr/aeiouAEIOU//);
```

- In this case, `$str` is not changed since the second `//` for `tr` is empty.

- There is a `/d` modifier to `tr` that will delete matched characters. To remove all spaces in a string you can use:

```
$str =~ tr/ //d;
```

- You could do the same thing with pattern substitution, but regular expression evaluation is slow, so if you can do without it, your code will run faster.

Finding repeated patterns using backreferences

- Suppose we had a file of SNP data and wanted to identify homozygous sites, e.g. AA, CC, GG, or TT. We could write:

```
if ($seq =~ /(AA|CC|GG|TT)/) {
    print "Found homozygous $1 \n";
}
```

- Remember, though, TIMTOWTDI. We could write instead:

```
if ($seq =~ /([ACGT])([ACGT])/ && $1 eq $2) {
    print "Found homozygous $1$2 \n";
}
```

- A more compact way to match this is to use the backreference \1 to refer to the first captured segment:

```
If ($seq =~ /([ACGT])\1/) {
    print "Found $1$1 in sequence\n";
}
```

- In a substitution, you can reference the first captured segment with `\1` within the first pair of `//` and `$1` within the second pair of `//` :

```
$str =~ s/($pat)\1/$1/;
```