Использование языка PERL для автоматизации работы с пространственными данными в среде Windows

Обсудить в форуме Комментариев — 0

Эта страница опубликована в основном списке статей сайта по адресу http://gis-lab.info/qa/perl.html

Как использовать язык PERL для автоматизации рутинных операций с данными с большим количеством примеров.

Содержание

Зачем это нужно

Данная статья безусловно не является пособием по языку PERL, однако она может помочь специалистам, активно использующим пространственные данные , в том числе данные дистанционного зондирования в своей работе. Увеличение количества доступных пространственных данных часто ставит перед исследователями задачу автоматизации процедур подготовки данных и их анализа. В большинстве случаев, при работе крупных информационных центров, эти задачи решаются написанием специальных программ и реализуются на платформах UNIX-LINUX. В тоже время, для большинства менее масштабных работ стандартных ГИС приложений и среды Windows обычно бывает достаточно для организации небольших потоков операций. Одним из средств организации таких потоков операций в Win32 является использование

языка PERL. Приведенные в статье примеры скриптов PERL могут не являться оптимальными с точки зрения программирования. Однако они позволяют решать поставленные задачи и просты для понимания и изменения пользователями.

Краткие основы языка PERL

О языке PERL

Язык PERL - Practical Extraction and Report Language (практический язык извлечений и отчетов) – был создан американским программистом Larry Wall для автоматизации работы с текстом и выполнения рутинных задач в операционной системе UNIX. В настоящее время (сентябрь 2006 года) стандартом является 5-ая версия интерпретатора языка, распространяемая свободно и доступная для платформ UNIX, LINUX, WIN32 и MAC OS. Примеры в этой статье используют версию ActivePerl для Windows (скачать дистрибутив ActivePerl). Основным удобством использования языка PERL в среде Windows является отсутствие необходимости в компилировании программ – Ваши скрипты будут автоматически запускать интерпретатор языка для выполнения, и изменение программы будет не сложнее внесение правок в текстовый файл! При установке интерпретатора проверьте, чтобы он был установлен в директорию по умолчанию - C:\Perl.

Для получения дополнительной информации и справки по языку можно использовать встроенную справочную систему (на английском), а также русскоязычные пособия, доступные в Интернет: Изучаем Perl (http://perl.find-info.ru/perl/016/index.htm), Введение в Perl (http://perl.org.ru/documentation/docs/perl/index.htm) и другие. Следует иметь ввиду, что как и во всяком OpenSource проекте, в PERL существует масса вариантов синтаксиса одних и тех же команд. В этой статье не всегда приводятся самые удобные и короткие выражения – Вы сможете заменить их на более простые при написании своих скриптов.

Типы данных, загрузка и вывод данных

В Perl используется три типа данных: переменные, массивы и хеши. В простых скриптах мы будем использовать только переменные и массивы. Имена переменных всегда начинаются с символа $, например: $filename PERL различает тип переменных по контексту данных, и нет необходимости специально декларировать список переменных в начале программы. В любом месте программы вы можете определить новую переменную и присвоить ей значение:

```
$filename=”mytext.txt”;
$number=20;
$newnumber=$oldnumber;
```

Массив представляет собой последовательность переменных любого типа, элементы массива нумеруются последовательно, начиная с 0. Массив обозначается символом @:

```
@alldata
```

Элемент массива определяется его номером:

```
#первый элемент массива
@alldata[0]
```

Вы можете присвоить переменной значение элемента массива:

```
$data=@alldata[0];
```

или определить переменную как размер (число элементов) в массиве:

```
$number=@alldata;
```

Поскольку мы будем рассматривать скрипты, оперирующие с текстами, то нам необходимо представлять себе как прочитать или записать данные в текстовый файл. Встроенной функцией open можно открыть файл для чтения:

```
open (DRR, "dirlist.txt");
```

или для записи:

```
open (OUT, "&gt;batch.bat");
```

Если файл с таким именем существует, он будет стерт и записан заново. После записи данных в файл его необходимо закрыть:

```
close (OUT);
```

Проверка значений, циклы

Основные функции языка программирования, которые нам понадобятся для написания скриптов, будут проверка значений и организация циклов процедур. Эти операции в PERL имеют схожий синтаксис:

```
#фигурная скобка обозначает начало блока операций
условие
{
операции
}
#конец блока операций
```

Проверка значений по типу «если да-то, если нет-то» выглядит так:

```
if ($p1&lt;=$p2)
{
$U=$p1;
}
else
{<br /> $U=$p2;
}
```

В данном примере переменной $U будет присвоено минимальное значение из двух вариантов.

Определение цикла операций выглядит следующим образом:

```
$num=10;
$inc=0;
while ($inc &lt; $num)
{
#набор операций
++$inc;
}
```

В данном случае цикл будет выполнен 10 раз, в конце каждого цикла мы увеличиваем значение переменной $inc на единицу - ++$inc.

Операторы, встроенные функции

Вот некоторые из операторов языка, которые будут нами использованы в примерах скриптов: Присвоение значения: «=» или «my … =»

```
#определение значения переменной
$inc=0;
#чтение содержимого файла, открытого под именем
#DRR  в массив @alldir
my @alldir=&lt;DRR&gt;;
```

Отношение: < (меньше), > (больше), <= (меньше или равно), == (равно – для чисел), != (не равно – для чисел), eq (равно – для строк), ne (не равно – для строк).

Арифметические операторы: +,-,*,/.

Мы также будем использовать встроенные функции, отвечающие за ввод и вывод данных и выполняющие простые операции над текстом:

open/close: открытие текстового файла для чтения или записи – open, и закрытие файла – close (синтаксис этих функция рассмотрен выше).

print: вывод на печать.

```
#значение переменной $line будет выведено на экран
print "$line";
#значение будет напечатано в текстовый файл, #открытый под именем OUT
print OUT "$line";
```

Следует отметить наиболее распространенные спецсимволы, которые можно выводить на печать: \n – конец строки, \t – символ табуляции, \ - символ пробела (пробел стоит после \), \символ - любой символ, если он зарезервирован во внутренней структуре PERL (например, для вывода символов “ ; \ необходимо набрать \“ \; \\ соответственно, иначе программа не будет работать!). split: разделяет строку по заданному символу-разделителю:

```
($filenam,$pnum,$zone)=split(',',$fileout);
```

Например: если $fileout = «image1,s_11,23» (строка таблицы с информацией о имени файла, номере точки и зоны проекции), то переменные будут иметь следующие значения:

```
$filenam=«image1»
$pnum=«s_11»
$zone=«23» (последняя переменная будет так же содержать символ переноса строки).
```

s///: оператор замены. В общем случае синтаксис оператора выгладит так:

переменная =~ s/символ-шаблон/символ для замены шаблона/;

Например, полученная выше переменная $zone содержит, кроме необходимой информации, также символ переноса строки. Что бы очистить его, используем замену:

$zone =~ s/\n//; - переменная $zone равна переменной $zone, в которой символ конца строки \n заменен на отсутствие символа.

system: выполняет системную функцию. Например, для получения списка файлов в директории, используем функцию MS-DOS dir:

```
$com="C:\\Geomatica_V91\\exe\\easi.exe r script";
system($com);
```

PERL запустит интерпретатор MS-DOS и выполнит указанную в переменной строку.

Общие правила написания скриптов в PERL и их отладка

Скрипт PERL в среде Windows представляет собой простой текстовый файл с расширением .pl. Если в Вашей системе установлен интерпретатор языка, то файлы *.pl будут иметь значок с желтым шариком, и будут выполняться автоматически по двойному щелчку на имя файла (подобно файлам .bat). Первая строка файла должна содержать путь к интерпретатору языка – в нашем случае строка эта будет выглядеть так:

```
#!/usr/local/bin/perl
```

Далее вводиться текст программы. Все строки, в которых используются операторы или функции, должны заканчиваться точкой с запятой - ;. Исключение составляют строки, определяющие условия или операторы циклов:

```
while ($inc &lt; $num)
{
```

```
print "привет";
++$inc;
}
```

Примечания обозначаются символом # - весь текст от символа до конца строки будет считаться примечанием.

Никаких специальных символов конца скрипта в PERL не используется. Для запуска скриптов, рассчитанных на работу со всеми файлами или поддиректориями в определенной директории, требуется скопировать файл скрипта в эту папку и запустить его. В противном случае Вам придется использовать не относительные, а абсолютные пути к папкам и файлам. Для отладки скриптов полезно бывает использовать интерпретатор MS-DOS. Для этого запустите Command Prompt, перейдите в директорию со скриптами, и наберите имя файла скрипта с расширением. Файл будет запущен, и информация об ошибках появиться на экране.

Примеры скриптов

Внимание: для работы с примерами, которые можно загрузить с данной страницы, необходимо сменить их расширение с *.txt на *.pl.

Автоматизация работы с файлами: Операции с файлами и директориями

Приведем для примера два простых скрипта. Первый скрипт создает поддиректории во всех поддиректориях рабочей директории (рабочей мы будем называть директорию, из которой запущен скрипт) (скачать скрипт):

```
#Стандартное начало скрипта:
#!/usr/local/bin/perl
#Определяем строковую переменную, содержащую команду DOS для вывода списка
поддиректорий в файл dirlist.txt (используем ключи команды dir: /A:D – только
директории, /O:N – сортировка по имени и /B – краткий формат) :
$com="dir /A:D /O:N /B &gt;dirlist.txt";
#Выполняем команду в операционной системе:
system($com);
#Открываем для чтения файл со списком поддиректорий:
open (DRR, "dirlist.txt");
#Читаем содержимое файла в массив @alldir:
my @alldir=&lt;DRR&gt;;
#Определяем длину массива:
$num=@alldir;
#Указываем номер первого элемента (напомним, что элементы в массиве нумеруются с нуля,
и последний элемент будет иметь номер на единицу меньший числа элементов):
$inc=0;
#Запускаем цикл операций, который будет продолжаться до тех пор, пока значение
переменной $inc (номер элемента массива) меньше переменной $num (число элементов в
массиве). В ходе этого цикла мы прочитаем последовательно все строки файла, цикл
закончиться после прочтения последней строки:
while ($inc &lt; $num)
#Начало операций цикла:
{
#Присваиваем переменной $line значение первой строки массива (то есть первой строки
текстового файла):
$line = @alldir[$inc];
#Удаляем ненужный символ конца строки:
$line =~ s/\n//;
#Определяем команду для создания поддиректории: команда DOS md (makedir), имя
поддиректории составляем из имени директории, сохраненного в переменной $line и нужного
повторяющегося имени, для вывода на печать символы \ и "  используем вспомогательный
символ \, показывающий, что стоящий за ним символ не является командой (поскольку эти
символы зарезервированы PERL). Имя директории указано в кавычках чтобы избежать проблем
с пробелами в именах директорий:
$com="md \"$line\\data\"\n";
#Выполняем команду в операционной системе:
system($com);
#Увеличиваем значение переменной $inc на единицу:
```

```
++$inc;
#Конец операций цикла:
}
#После выполнения команд удалим файл dirlist.txt. Для этого сначала закроем его:
close (DRR);
#Затем определим команду DOS:
$com="del dirlist.txt\n";
#И выполним ее:
system($com);
```

Примечание – хотя данный скрипт может использовать директории с пробелами, многие приведенные ниже скрипты при наличии пробелов в именах директорий и файлов не будут работать. Впрочем, тоже относиться и к большинству распространенных ГИС-приложений.

Второй скрипт скопирует файлы с именем time1.tif из всех поддиректорий рабочей директории в директорию output, заменив имя time1 на имя поддиректории, из которой он был скопирован (то есть файл time1.tif из директории 123_056 получит имя 123_056.tif и так далее). Мы могли бы использовать такой же подход, как в предыдущем случае, но сознательно применили другой путь – теперь скрипт PERL не будет самостоятельно выполнять команды ОС, а создаст исполняемый файл, который можно будет запустить позже. Хотя такой подход менее продуктивен, он позволяет избежать необходимости восстанавливать данные при ошибках в программе – понятно, что исполняемый batch файл легче проверить и убедиться, что он будет работать правильно (скачать скрипт).

```
#!/usr/local/bin/perl
#Создаем список директорий и читаем его:
$com="dir /A:D /O:N /B &gt;dirlist.txt";
system ($com);
open (DRR, "dirlist.txt");
#Открываем файл для записи (если файл batch.bat существует, он будет стерт и
перезаписан):
open (OUT, "&gt;batch.bat");
#Печатаем в batch команду создания директории output:
print OUT "md output\n";
#Запускаем цикл операций:
my @alldir=&lt;DRR&gt;;
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
#Читаем имена директорий:
$line = @alldir[$inc];
$line =~ s/\n//;
#Печатаем команду копирования и переименования файла:
print OUT "copy \"$line\\time1.tif\" \"output\\$line.tif\"\n";
++$inc;
}
#Закрываем файлы:
close (OUT);
close (DRR);
#Удаляем файл dirlist.txt:
$com="del dirlist.txt\n";
system($com);
```

Автоматизация работы с файлами: Распаковка архивов

Рассмотрим примеры, более близкие к практике. Скрипт, приведенный ниже, может помочь в автоматической распаковке снимков Landsat-7, скаченных с сайта GLCF (http://glcfapp.umiacs.umd.edu:8080/esdi/index.jsp). Мы скачали целиком директории со снимками и теперь хотим удалить все технические файлы и ненужные каналы, и распаковать файлы tif из архивов GZ с помощью архиватора WinRAR (установленного в C:\WinRAR) (скачать скрипт):

```
#!/usr/local/bin/perl
#Создаем список директорий и читаем его:
$com="dir /A:D /O:N /B &gt;dirlist.txt";
```

```
system($com);
open (DRR, "dirlist.txt");
#Открываем файл для записи:
open(OUT, "&gt;batch.bat");
#Запускаем цикл операций:
my @alldir=&lt;DRR&gt;;
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
#Читаем имена директорий:
$line = @alldir[$inc];
$line =~ s/\n//;
#Печатаем команду перехода в поддиректорию:
print OUT "cd $line\n";
#Печатаем команды удаления ненужных файлов (если файлы отсутствуют, команды будут
пропущены):
print OUT "del /Q *_nn6*\n";
print OUT "del /Q *_nn8*\n";
print OUT "del /Q *_nn1*\n";
print OUT "del /Q *.jpg\n";
print OUT "del /Q *.met\n";
print OUT "del /Q *.hdr\n";
print OUT "del /Q *.i*\n";
#Печатаем команду разархивирования:
print OUT "C:\\WinRAR\\winrar.exe e -ibck *.gz\n";
#Печатаем команду удаления архивов:
print OUT "del /Q *.gz\n";
#Печатаем команду возврата в рабочую директорию:
print OUT "cd ..\n";
++$inc;
}
close (DRR);
close (OUT);
$com="del dirlist.txt";
system($com);
```

Автоматизация работы с файлами: Упаковка файлов

Если вы получили снимки Landsat из архивов EROS Data Center, то, скорее всего, они будут в формате tif без сжатия. Мы хотели бы сжать эти файлы в архивы формата GZ с помощью архиватора GZip (установленного в C:\Gzip), причем каждый файл tif в отдельный архивный файл. Для этого мы используем вот такой скрипт:

```
#!/usr/local/bin/perl
#Создаем список директорий и читаем его:
$com="dir /A:D /O:N /B &gt;dirlist.txt";
system($com);
open (DRR, "dirlist.txt");
#Запускаем цикл операций:
my @alldir=&lt;DRR&gt;;
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
#Читаем имена директорий:
$line = @alldir[$inc];
$line =~ s/\n//;
#Создаем список файлов в поддиректории  и читаем его:
$com="dir $line\\*.tif /B &gt;dirlist$inc.txt\n";
system($com);
$newname="dirlist$inc.txt";
open (FRR, $newname);
#Запускаем второй цикл операций внутри первого цикла:
my @files=&lt;FRR&gt;;
$num1=@files;
```

```
$inc1=0;
while ($inc1 &lt; $num1)
{
#Читаем имена файлов:
$line1 = @files[$inc1];
$line1 =~ s/.tif\n//;
#Определяем и выполняем команду упаковки файлов:
$com = "C:\\Gzip\\gzip $line\\$line1.tif\n";
system($com);
++$inc1;
#Завершаем второй цикл:
}
++$inc;
Завершаем первый цикл:
}
close (DRR);
close (FRR);
#Удаляем все временные текстовые файлы:
$com="del dirlist*.txt\n";
system($com);
```

Автоматизация работы в PCI Geomatica: Запуск скриптов EASI

Одним из наиболее удобных приложений PCI Geomatica является язык EASI – простой язык для написания скриптов, позволяющих выполнить большинство распространенных операций подготовки и анализа данных ДЗ. Скрипты EASI имеют расширение .eas и могут быть запущены из командной строки DOS – имя скрипта указывается в качестве атрибута при вызове программы. Простейший скрипт PERL, приведенный ниже, позволяет запускать один и тот же скрипт EASI последовательно в каждой поддиректории рабочей директории – то есть позволяет выполнить однотипные операции над всеми наборами данных, используемых в Вашем проекте. Для удобства использования все необходимые скрипты EASI заранее собраны в директории C:\scripts\, так что перед выполнением скрипта необходимо только указать имя нужного скрипта для исполнения (скачать скрипт):

```
#!/usr/local/bin/perl
$com1="dir /A:D /O:N /B &gt;dirlist.txt";
system($com1);
open (DRR, "dirlist.txt");
open(OUT, "&gt;batch.bat");
my @alldir=&lt;DRR&gt;;
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
$line = @alldir[$inc];
$line =~ s/\n//;
#Печатаем комманды в исполняемый файл DOS: переходим в поддиректорию...
print OUT "cd $line\n";
#...копируем скрипт, переименовывая его в script.eas...
 print OUT "copy C:\\scripts\\myprocess.eas script.eas /y\n";
#...выполняем скрипт...
print OUT "C:\\Geomatica_V91\\exe\\easi.exe r script\n\n";
#...и удаляем скрипт и файл параметров EASI:
print OUT "del script.eas\n\n";
print OUT "del prm.prm\n\n";
#Возвращаемся в рабочую директорию:
print OUT "cd ..\n";
++$inc;
}
close (OUT);
close (DRR);
$com="del dirlist.txt";
system($com);
```

Автоматизация работы в PCI Geomatica: Генерирование скриптов EASI

Простота языка EASI делает возможным автоматическое создание скриптов с помощью программы PERL по заданному шаблону. Поскольку задачи, для которых может потребоваться подобная процедура, достаточно индивидуальны, мы не будем приводить примера готового скрипта, а покажем только часть кода. В данной задаче исходными данными служили файлы с таблицами координат углов прямоугольника, спроецированного из синусоидальной проекции в UTM. Требовалось построить пустую растровую базу данных PCIDSK с размером ячейки 28.5 метров таким образом, что бы она полностью включала в себя полученный прямоугольник. Данный кусок кода представляет собой операцию внутри цикла:

```
#Вне цикла мы отрыли  список файлов (в формате: «имя файла, номер полигона, номер зоны UTM») и  прочитали его в массив @filelist. В ходе цикла мы открываем строку файла списка,  читаем ее, выделяем имя файла и открываем этот файл:
$fileout = @filelist[$inc];
($filenam,$pnum,$zone)=split(',',$fileout);
$zone =~ s/\n//;
open (OUT, "$filenam");
#Читаем содержимое  файла – четыре строки с парами координат:
my @filetext = &lt;OUT&gt;;
$line1 = @filetext[0];
$line2 = @filetext[1];
$line3 = @filetext[2];
$line4 = @filetext[3];
#Разбиваем строки с  получением значений координат:
($p1_x,$p1_y)=split(' ',$line1);
($p2_x,$p2_y)=split(' ',$line2);
($p3_x,$p3_y)=split(' ',$line3);
($p4_x,$p4_y)=split(' ',$line4);
#Сравниваем координаты  для нахождения углов прямоугольника, полностью включающего в себя исходный  прямоугольник (extent):
if ($p1_x&lt;=$p3_x)
{
$UL_x=$p1_x;
}
else
{
$UL_x=$p3_x;
}
if ($p3_y&lt;=$p2_y)
{
$UL_y=$p2_y;
}
else
{
$UL_y=$p3_y;
}
if ($p2_x&lt;=$p4_x)
{
$LR_x=$p4_x;
}
else
{
$LR_x=$p2_x;
}
if ($p1_y&lt;=$p4_y)
{
$LR_y=$p1_y;
}
else
{
$LR_y=$p4_y;
}
#Создаем новую  директорию с использованием функции PERL:
mkdir "s_$pnum";
#Расширяем размер  создаваемой базы данных на 100 метров для избегания краевых эффектов при  последующих операциях:
```

```
$ULx2=$UL_x-100;
$ULy2=$UL_y+100;
$LRx2=$LR_x+100;
$LRy2=$LR_y-100;
#Определяем размер  создаваемой БД в пикселях:
$sizeX2=int(($LRx2-$ULx2)/28.5);
$sizeY2=int(($ULy2-$LRy2)/28.5);
#Рассчитываем новые координаты  правого нижнего угола исходя из размера пикселя:
$LRx2n=$ULx2+($sizeX2*28.5);
$LRy2n=$ULy2-($sizeY2*28.5);
#Печатаем команды  создания базы данных (файл с именем EASI заранее был открыт для
записи):
print EASI "FILE=\"s_$pnum\\clip\n";
print EASI "DBSZ=$sizeX2,$sizeY2\n";
print EASI "DBNC=15,0,0,0\n";
print EASI "DBLAYOUT=\"FILE\n";
print EASI "RUN CIM\n";
print EASI "\n";
#Печатаем команды  присвоения новой БД параметров проекции. Обратите внимание на
использование  переменной $zone, которая была прочитана в самом начале цикла.
print EASI "UPLEFT=$ULx2,$ULy2\n";
print EASI "LORIGHT=$LRx2n,$LRy2n\n";
print EASI "MAPUNITS=\"UTM $zone E012\n";
print EASI "RUN GEOSET\n";
print EASI "\n";
```

Примечание – для включения скриптов EASI в непрерывный процесс Вы можете запускать созданные скрипты автоматически из PERL, используя команду:

```
$com="C:\\Geomatica_V91\\exe\\easi.exe r script";
system($com);
```

После чего использовать результаты выполнения скрипта для дальнейших действий.

Автоматизация работы в ERDAS Imagine: Генерирование пакетов команд

Скрипт, приведенный в первом примере, создаст пакетный файл команд ERDAS Imagine для относительно простой процедуры Layer merge. В параметрах процедуры требуется указать абсолютные имена файлов. Для этого нам необходимо знать или задать имя рабочей директории. Хотя определить имя директории, из которой запущен скрипт, можно автоматически, мы, в данном случае, использовали переменную, значение которой необходимо изменить на абсолютный путь к рабочей директории перед запуском скрипта. Так же обратите внимание на использование в ERDAS путей в стандарте UNIX. Данный скрипт рассчитан на определенный шаблон имени файлов (стандарт Ortho Landsat для GLCF): p169r057_7t20000127_z37_nn30.tif, и будет объединять только 3,4 и 5 каналы снимков (скачать скрипт):

```
#!/usr/local/bin/perl
#Абсолютный путь к рабочей директории (обратите внимание на
#использование стандарта UNIX в путях!):
$path="D:/glcf";
#Создание списка папок и запуск цикла:
$com="dir /A:D /O:N /B &gt;dirlist.txt";
system($com);
open (DRR, "dirlist.txt");
open(OUT, "&gt;merge.bcf");
my @alldir=&lt;DRR&gt;;
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
$line = @alldir[$inc];
$line =~ s/\n//;
#Печатаем в файл имя одного из файлов tif из папки для получения шаблона
```

```
#имени:
$com="dir $line\\*nn20.tif /B &gt;dirlist$inc.txt\n";
system($com);
$newname="dirlist$inc.txt";
open (FRR, $newname);
my @files=&lt;FRR&gt;;
$line1 = @files[0];
#Разбиваем имя - получаем позицию сцены, дату и номер зоны UTM:
($wrs,$date,$temp1,$temp2)=split('\_',$line1);
#Печатаем длинную строку команды layermerge в пакетный файл ERDAS bcf.
#Обратите внимание, что это одна строка, переносы здесь недопустимы!
print OUT "modeler -nq \$IMAGINE_HOME/etc/models/layermerge.pmdl -meter -state Union
None \'FLOAT RASTER n1 FILE OLD NEAREST NEIGHBOR AOI NONE EDGE
FILL\"$path/$line/$wrs\_$date\_$temp1\_nn50.tif\"\; FLOAT RASTER n2 FILE OLD NEAREST
NEIGHBOR AOI NONE EDGE FILL\"$path/$line/$wrs\_$date\_$temp1\_nn40.tif\"\; FLOAT RASTER
n3 FILE OLD NEAREST NEIGHBOR AOI NONE EDGE
FILL\"$path/$line/$wrs\_$date\_$temp1\_nn30.tif\"\;' ignore Unsigned_8_bit
\"$path/$wrs\_$date\_utm.img\" \'n1(1)\,n2(1)\,n3(1)\'\n";
++$inc;
}
close (OUT);
close (DRR);
$com="del dirlist*.txt";
system($com);
```

Второй пример содержит скрипт для расчета среднего и стандартного отклонения в плавающем окне для всех файлов stat.tif из всех поддиректорий рабочей директории. Его основным отличием от предыдущего скрипта является необходимость внесения путей к файлам в отдельные переменные. Смена абсолютного пути к рабочей директории здесь не продумана, так что при необходимости изменения, придется изменить все четыре пути к файлам (более правильным было бы введение дополнительной переменной, как в предыдущем варианте).

```
#!/usr/local/bin/perl
$com="dir /A:D /O:N /B &gt;dirlist.txt";
system($com);
open (DRR, "dirlist.txt");
open(OUT, "&gt;batch.bcf");
my @alldir=&lt;DRR&gt;;
$num=@alldir;
#Поскольку мы будем создавать два набора переменных для каждого файла,
#увеличиваем число шагов вдвое:
$num=$num*2;
$inc=0;
while ($inc &lt; $num)
{
$n=$inc/2;
$line = @alldir[$n];
$line =~ s/\n//;
#Печатаем первую команду - вычисление среднего:
print OUT "variable Inp$inc Auto \"d:/data/$line/stat.tif\"\;\n";
print OUT "variable Out$inc Auto \"d:/data/$line/statmean.img\" Delete_Before\;\n";
<nowiki> print OUT "modeler -nq \$IMAGINE_HOME/etc/models/focalanal.pmdl -meter -state
\"\$(Inp$inc)\" Integer \"\$(Out$inc)\" 'Unsigned 8 bit' Float 'matrix (3,3:
1.000,1.000,1.000,1.000,1.000,1.000,1.000,1.000,1.000)' Mean '\$(Inp$inc.Ulx)'
'\$(Inp$inc.Uly)' '\$(Inp$inc.Lrx)' '\$(Inp$inc.Lry)' Map useall \"None\" '' '' ''
''\n";</nowiki>
++$inc;
#Вторая команда - вычисление стандартного отклонения:
print OUT "variable Inp$inc Auto \"d:/data/$line/stat.tif\"\;\n";
print OUT "variable Out$inc Auto \"d:/data/$line/statstd.img\" Delete_Before\;\n";
<nowiki> print OUT "modeler -nq \$IMAGINE_HOME/etc/models/focalanal.pmdl -meter -state
\"\$(Inp$inc)\" Integer \"\$(Out$inc)\" 'Float Single' Float 'matrix (3,3:
1.000,1.000,1.000,1.000,1.000,1.000,1.000,1.000,1.000)' SD '\$(Inp$inc.Ulx)'
'\$(Inp$inc.Uly)' '\$(Inp$inc.Lrx)' '\$(Inp$inc.Lry)' Map useall \"None\" '' '' ''
```

```
''\n";</nowiki>
++$inc;
}
close (OUT);
close (DRR);
$com="del dirlist.txt";
system($com);
```

Автоматизация работы в ERDAS Imagine: Генерирование моделей

Однако иногда встроенных команд ERDAS бывает недостаточно и возникает необходимость запуска собственных моделей. В таком случае мы использовали следующий подход: сначала создается и тестируется модель, потом она записывается в файл в виде скрипта, затем команды скрипта переносятся в PERL в качестве атрибутов функции print. В таком случае, для каждого набора файлов PERL будет создавать собственную модель, используя абсолютные пути к файлам, а затем все созданные модели будут включены в пакетный файл для автоматического запуска. Ниже приведен пример скрипта для создания и выполнения моделей для расчета стандартного отклонения для 5-канальных файлов stat.tif из всех поддиректорий (скачать скрипт):

```
#!/usr/local/bin/perl
$com="dir /A:D /O:N /B &gt;dirlist.txt";
system($com);
open (DRR, "dirlist.txt");
#Открываем для записи пакетный файл:
open(OUT, "&gt;batch.bcf");
my @alldir=&lt;DRR&gt;;
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
$line = @alldir[$inc];
$line =~ s/\n//;
#Создаем и открываем файл модели:
open(MOD, "&gt;std$line.mdl");
#Печатаем команды модели для данного набора файлов (нет переносов!):
print MOD "COMMENT \"no comment\";\n";
print MOD "SET CELLSIZE MIN\;\n";
print MOD "SET WINDOW UNION\;\n";
print MOD "SET AOI NONE\;\n";
print MOD "Integer RASTER n1_temp FILE OLD NEAREST NEIGHBOR AOI NONE
\"D\:/data/$line/stat.tif\"\;\n";
print MOD "Integer RASTER n11_std FILE NEW USEALL ATHEMATIC 8 BIT UNSIGNED INTEGER
\"D\:/data/$line/statstd1.img\"\;\n";
print MOD "Integer RASTER n12_std FILE NEW USEALL ATHEMATIC 8 BIT UNSIGNED INTEGER
\"D\:/data/$line/statstd2.img\"\;\n";
print MOD "Integer RASTER n13_std FILE NEW USEALL ATHEMATIC 8 BIT UNSIGNED INTEGER
\"D\:/data/$line/statstd3.img\"\;\n";
print MOD "Integer RASTER n14_std FILE NEW USEALL ATHEMATIC 8 BIT UNSIGNED INTEGER
\"D\:/data/$line/statstd4.img\"\;\n";
print MOD "Integer RASTER n15_std FILE NEW USEALL ATHEMATIC 8 BIT UNSIGNED INTEGER
\"D\:/data/$line/statstd5.img\"\;\n";
print MOD "INTEGER MATRIX n6_Low_Pass\;\n";
print MOD "n6_Low_Pass = MATRIX(3, 3\:\n";
print MOD "\t1, 1, 1,\n";
print MOD "\t1, 1, 1, \n";
print MOD "\t1, 1, 1)\;\n";
print MOD "n11_std = FOCAL STANDARD DEVIATION ( \$n1_temp(1) , \$n6_Low_Pass ) \;\n";
print MOD "n12_std = FOCAL STANDARD DEVIATION ( \$n1_temp(2) , \$n6_Low_Pass ) \;\n";
print MOD "n13_std = FOCAL STANDARD DEVIATION ( \$n1_temp(3) , \$n6_Low_Pass ) \;\n";
print MOD "n14_std = FOCAL STANDARD DEVIATION ( \$n1_temp(4) , \$n6_Low_Pass ) \;\n";
print MOD "n15_std = FOCAL STANDARD DEVIATION ( \$n1_temp(5) , \$n6_Low_Pass ) \;\n";
print MOD "QUIT\;\n";
#Закрываем файл модели:
close (MOD);
#Печатаем команду выполнения модели в пакетный файл:
print OUT "modeler -nq D:/data/std$line.mdl -meter -state\n";
```

```
++$inc;
}
close (OUT);
close (DRR);
```

*Примечание – данный скрипт является примером организации работы с пользовательскими моделями в ERDAS Imagine. Возможны более продуктивные подходы – например, написание моделей с пользовательскими переменными и пр.

Автоматизация работы в ArcInfo: Генерирование скриптов AML

Разумеется, PERL имеет огромный потенциал для автоматического создания скриптов AML. К сожалению, автор практически не использовал в своей работе ArcInfo. В качестве примера можно использовать небольшой скрипт PERL для создания огромного файла для выполнения команды Generate.

Автоматизация импорта данных MODIS: Использование MRT

Использование пакетных файлов для запуска MRT уже реализовано в виде, значительно более дружелюбном к пользователю, чем использование скриптов PERL – например, существует специальная программа, modisimport-tool.html представленная на сайте GIS-Lab. Тем не менее, мы приведем здесь один из вариантов скриптов, который может быть модифицирован и является общим (хотя и не оптимальным) решением проблемы автоматического импорта, склеивания и перепроецирования продуктов MODIS. Для использования скрипта Вам потребуется изменить путь к программе MRT на тот, который Вы использовали при установке, и разместить свои данные в директории C:\modwork\input. В случае данного скрипта используются 6 ячеек сетки для каждого композита – если число ячеек другое, его надо изменить, также надо изменить параметры проекции и набор каналов. Программа создаст список файлов HDF в Вашей директории, создаст папки для мозаик (C:\modwork\mosaic) и для выгрузки данных (C:\modwork\output), файлы с параметрами перепроецирования для каждого набора мозаик и исполняемый файл для запуска мозаики и перепроецирования (скачать скрипт).

Автоматизация импорта данных MODIS: Использование MRT-Swath

Более важная для нас операция – автоматический импорт и перепроецирование продуктов MOD02QKM (Swath продуктов MODIS) с помощью программы MRT-Swath еще не решен в виде отдельного приложения, и поэтому использование скрипта PERL здесь оправдано. Для запуска скрипта надо изменить (или переместить данные):

1. Имя папки с данными – по умолчанию файлы MOD02QKM находятся в D:\MODISWORK\MOD02QKM, а геолокационные поля к ним (MOD03) – в папке D:\\MODISWORK\MOD03\.
2. Путь к программе – в данной версии это C:\MRTSwath\.
3. Параметры проекции и набор каналов. Мы предполагаем, что для всех сцен скачены оба продукта (MOD02QKM и MOD03), и их наборы полностью соотвествуют друг другу. Скрипт создаст папку для выгрузки результатов - D:\MODISWORK\output\, исполняемый файл batch.bat для запуска перепроецирования и набор файлов параметров для каждого снимка. Запускать скрипт надо из директории D:\MODISWORK\ (или той, на которую Вы измените это название!):

```
#!/usr/local/bin/perl
#Создаем список продуктов MOD02QKM:
$com="dir MOD02QKM\\*.hdf /B &gt;dir1.txt";
system($com);
#Создаем список продуктов MOD03 (мы предполагаем, что для всех снимков
#скачены геолокационные поля):
$com="dir MOD03\\*.hdf /B &gt;dir2.txt";
system($com);
#Создаем директорию для выгрузки данных
$com="md output";
system($com);
#Читаем файлы и создаем исполняемый файл:
open (DR1, "dir1.txt");
open (DR2, "dir2.txt");
open(OUT, "&gt;batch.bat");
```

```
my @allmodis=&lt;DR1&gt;;
my @allgeo=&lt;DR2&gt;;
#Запускаем цикл для всех снимков:
$num=@allmodis;
$inc=0;
while ($inc &lt; $num)
{
#Читаем имя файла
$line1 = @allmodis[$inc];
$line2 = @allgeo[$inc];
#Разбиваем имя по шаблону и создаем имя выходного файла:
($temp1,$temp2,$temp3,$temp4,$temp5,$temp6)=split('\.',$line1);
$newfile="$temp1\_$temp2\_$temp3.tif";
#Создаем файл параметров:
open(PRM, "&gt;mod$inc.prm");
#Печатаем имена файлов для импорта:
print PRM "INPUT_FILENAME = D:\\MODISWORK\\MOD02QKM\\$line1\n";
print PRM "GEOLOCATION_FILENAME = D:\\MODISWORK\\MOD03\\$line2\n";
#Печатаем список каналов:
print PRM "INPUT_SDS_NAME = EV_250_RefSB, 0, 1\n";
#Печатаем имя выходного файла:
print PRM "OUTPUT_FILENAME = D:\\MODISWORK\\output\\$newfile\n";
#Печатаем формат и параметры проекции выходного файла:
print PRM "OUTPUT_FILE_FORMAT = GEOTIFF_FMT\n";
print PRM "KERNEL_TYPE (CC/BI/NN) = NN\n";
print PRM "OUTPUT_PROJECTION_NUMBER = ALBERS\n";
print PRM "OUTPUT_PROJECTION_PARAMETER = 0.0 0.0 52.0 64.0 45.0 0.0 8500000.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0\n";
print PRM "OUTPUT_PROJECTION_SPHERE = 15\n";
print PRM "OUTPUT_PIXEL_SIZE = 250\n";
#Закрываем файл параметров:
close (PRM);
#Печатаем в командный файл команду импорта и перепроецирования:
print OUT "C:\\MRTSwath\\bin\\swath2grid -pf=mod$inc.prm\n";
++$inc;
}
close (OUT);
close (DRR);
close (FRR);
$com="del *.txt\n";
system($com);
```

Автоматизация обработки текстовых отчетов

Обработка текстов и генерирование отчетов является основной задачей PERL, однако при использовании в ГИС такие задачи не являются первостепенными. Тем не менее, при получении большого количества результатов, для их сравнения иногда необходимо применять автоматизацию. Например, после классификации большого количества снимков требуется создание суммарной таблицы соотношения классов. В нашем случае каждый снимок классифицировался на классы «лес» и «не лес», кроме того, ряд пикселей был отмечен как «нет данных». Таким образом, результирующий растровый слой изображения содержит три класса данных: 1 – нет данных 2 – лес 3 - не лес Мы можем выгрузить гистограмму значений этого слоя в текстовый файл (например, командой EASI histdump) – таким образом, в каждой директории с данными мы создадим файл blockforest.txt, который представляет собой колонку чисел: 10 – число пикселей первого класса 128779 – число пикселей второго класса и т.д.... 6576 0 0 ... Создадим скрипт для автоматического сбора данных из всех файлов blockforest.txt и создания таблицы:

```
#!/usr/local/bin/perl
#Создадим список поддиректорий для обработки:
$com="dir /A:D /O:N /B &gt;dirlist.txt";
system($com);
#Откроем список директорий и создадим файл с результирующей таблицей:
open (DRR, "dirlist.txt");
open(OUT, "&gt;output.txt");
#В результирующую таблицу добавим «шапку»:
```

```
print OUT "name\t nodata\t forest\t noforest\n";
my @alldir=&lt;DRR&gt;;
#Запустим цикл:
$num=@alldir;
$inc=0;
while ($inc &lt; $num)
{
#Читаем имя директории:
$line = @alldir[$inc];
$line =~ s/\n//;
#Открываем текстовый фал из нужной директории:
$fileforest="$line\\blockforest.txt";
open (bforest, "$fileforest");
#Читаем значения гистограммы:
my @allforest=&lt;bforest&gt;;
$bforest1 = @allforest[0];
$bforest1 =~ s/\n//;
$bforest2 = @allforest[1];
$bforest2 =~ s/\n//;
$bforest3 = @allforest[2];
$bforest3 =~ s/\n//;
#Печатаем значения переменных в результирующую таблицу:
print OUT "$line\t$bforest1\t$bforest2\t$bforest3\n";
++$inc;
}
close (OUT);
close (DRR);
$com="del dirlist.txt";
system($com);
```

Мы не будем предлагать этот скрипт для загрузки, поскольку все операции по анализу текстовых отчетов являются индивидуальными. Отметим, что с помощью PERL возможно создание программ для анализа весьма сложных текстовых конструкций. В тех случаях, когда количество переменных велико или изначально неизвестно, можно, для облегчения задачи, создавать промежуточные текстовые файлы, которые затем будут анализироваться (например, выгружать гистограммы по каждому классу в отдельные файлы в виде столбцов данных для последующего статистического анализа). Метод создания промежуточных текстовых файлов является неоптимальным с точки зрения программиста, однако он очень прост для начинающего пользователя и позволяет проверить все промежуточные результаты при необходимости.

Ссылки по теме

- Perl.org

Обсудить в форуме Комментариев — 0

Последнее обновление: 2014-05-15 01:37

Дата создания: 10.11.2006
Автор(ы): Пётр Потапов