GSE 760–S01

Advanced Methods in Geospatial Modeling:

Computation for Remote Sensing Analysis and Product Generation

xiaoyang Zhang, 2/26/2021

Class Schedule

Date	Lecture (Friday)	Date	Exercises (Friday)
Jan 15	**Lecture 1**: Course overview and introduction to remote sensing processing on Linux system	Jan 15	Lab assignment 1- Linux system setup
Jan 22	**Lecture 2**: Getting start with Linux system	Jan 25	Lab assignment 2 - Command line syntax
Jan 29	**Lecture 3:** Linux files and file utilities	Jan 29	Lab assignment 3 – File utilities
Feb 5	**Lecture 4:** File system and processes	Feb 5	Lab assignment 4 – File system and processes
Feb 12	**Lecture 5:** Shell scripting	Feb 12	Lab assignment 5- Shell scripting
Feb 19	**Lecture 6:** Perl scripting (1)	Feb 19	Lab assignment 6- Perl scripting
Feb 26	**Lecture 7:** Perl scripting (2)	Feb 26	Lab assignment 7- Perl scripting
March 5	**Lecture 8:** Python scripting	March 5	Midterm Exam
March 12	**Spring Break**		**Spring Break**
March 19	**Lecture 9:** Satellite data and file format	March 19	Lab assignment 8- Python scripting
March 26	**Lecture 10:** Satellite data processing	March 26	Lab assignment 9- Satellite data processing
April 2	**No class/Easter Recess**	April 2	**No class/Easter Recess**
April 9	**Lecture 11:** Operational product generation	April 9	Lab assignment 10- Programing for product generation
April 16	**Lecture 12:** Software and product documentation	April 16	Project work overview
April 23	Work on projects	April 23	Work on projects
April 30	Work on projects	April 30	Work on projects
May 7	**Lecture 13:** Final presentation		

Final Exam: We will schedule final presentations during the final exam period.

Note: Recommended to readings to accompany each chapter will be assigned on the class D2L site.

Perl Scripting for Data Processing

xiaoyang Zhang, 2/26/2021

Review

Scalar Variables
Array
Hash
Operators
Conditional Statements and Looping

Subroutines and Functions

Generate a subroutine:

```
sub subroutine_name {
body of the subroutine
 }
```

Call a subroutine:

```
subroutine_name( list of arguments );
```

Subroutines and Functions

Lect8_1.pl

```
#!/usr/bin/perl
$num = Average(10, 20, 30);   #function
print "Average for the given numbers:  $num\n";
# Function definition
sub Average{
# get total number of arguments passed.
$n = scalar(@_);
$sum = 0;
foreach $item (@_)
{
$sum += $item;
 }
$average = $sum / $n;
return $average;
}
# Function call
```

Subroutines and Functions

Lect8_2.pl

```
#!/usr/bin/perl
# Function definition
@x=(10,20,30);
&Average;   # Function/subroutine call
print "Average for the given numbers:  $average\n";

sub Average{
# get total number of arguments passed.
 my $n = scalar(@x);
$sum = 0;
foreach $item (@x)
{
$sum += $item;
}
$average = $sum / $n;
return $average;
}
```

Subroutines

```
$n = &max(10, 15);
# This sub call has two parameters
```

```
sub max {
if ($_[0] > $_[1])
{
$_[0];
}
else {
$_[1];
}
}
```

```
my $answer = prompt();
# some code
my $second_answer = prompt();

sub prompt {
print "Have we arrived already?";
my $answer = <STDIN>;
chomp $answer;
return $answer;
}
```

Chomp function

Chomp is a safer version of chop that removes any trailing string that corresponds to the current value of $/. It returns the total number of characters removed from all its arguments. (By default , $/ is set to new line character.)

Lect8_3.pl

```
#!/usr/bin/perl
$string1 = "This is test";
$retval = chomp( $string1 );
print " Chopped String is : $string1\n";
print " Number of characters removed : $retval\n";
$string1 = “This is test\n";
$retval = chomp( $string1 );
print " Chopped String is : $string1\n";
print " Number of characters removed : $retval\n";
```

```
Chopped String is : This is test Number of characters removed : 0
Chopped String is : This is test Number of characters removed : 1
```

Time and gmtime

"time" function returns the number of seconds since the epoch (00:00:00 UTC, January 1, 1970, for most systems). Suitable for feeding to gmtime and localtime.

(seconds, minutes, hours, day of month, month, year, day of week, day of year, daylight savings time)

Lect8_4.pl

```
#!/usr/bin/perl -w
@weekday = ("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
$retval = time();
print "Return time is $retval\n";
$local_time = gmtime( $retval);
print "Local time = $local_time\n";
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = gmtime(time);
$year = $year + 1900;
$mon += 1;
print "Formated time = $mday/$mon/$year $hour:$min:$sec  $weekday[$wday]\n";
```

```
Return time is 1176831539
Local time = Fri Feb 22 17:38:59 2019
Formatted time = 22/02/2019 17:38:59 Fri
```

File Statistics and stat()

Inspect file attributes with the **stat()** function.

Function stat returns ARRAY, ($device, $inode, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime, $ctime, $blksize, $blocks)

```
 0 dev      device number of filesystem
 1 ino      inode number
 2 mode     file mode  (type and permissions)
 3 nlink    number of (hard) links to the file
 4 uid      numeric user ID of file's owner
 5 gid      numeric group ID of file's owner
 6 rdev     the device identifier (special files only)
 7 size     total size of file, in bytes
 8 atime    last access time in seconds since the epoch
 9 mtime    last modify time in seconds since the epoch
10 ctime    inode change time in seconds since the epoch (*)
11 blksize  preferred I/O size in bytes for interacting with the
            file (may vary from file to file)
12 blocks   actual number of system-specific blocks allocated
            on disk (often, but not always, 512 bytes each)
```

File Statistics and stat()

Lect8_5.pl

```
#!/usr/bin/perl -w
($device, $inode, $mode, $nlink, $uid, $gid, $rdev, $size,
$atime, $mtime, $ctime, $blksize, $blocks) =
stat("/etc/passwd");

print("stat() $device, $inode, $ctime\n");
```

```
stat() 147, 20212116, 1177094582
```

```
#!/usr/bin/perl -w
@info = stat (“filename.txt”);
$size=(stat (“filename.txt”))[7];
print(“device=$info[0], inode=$info[1], ctime=$info[10], size \n");
```

Perl File I/O

Open files:

open FILEHANDLE, EXPR

open FILEHANDLE

sysopen FILEHANDLE, FILENAME, MODE, PERMS

sysopen FILEHANDLE, FILENAME, MODE

Entities	Examples	Definition
< or r	open(DATA, "<file.txt");	Read Only Access
> or w	open(DATA, “>file.txt");	Creates, Writes, and Truncates
>> or a	open(DATA, “>>file.txt");	Writes, Appends, and Creates
+< or r+	open(DATA, “+<file.txt");	Reads and Writes
+> or w+	open(DATA, “+>file.txt");	Reads, Writes, Creates, and Truncates
+>> or a+	open(DATA, “+>>file.txt");	Reads, Writes, Appends, and Creates

Close files:

close FILEHANDLE

close (FILEHANDLE)

File I/O

```
open(my $fh, "<", "input.txt")
or die "cannot open < input.txt: $!";
```

($!--yields the corresponding system error string)

```
open(my $fh, ">", "output.txt")
or die "cannot open > output.txt: $!";
```

If **FILEHANDLE** is an **undefined** scalar variable (or array or hash element), a new **filehandle** is auto-vivified, meaning that the variable is assigned a reference to a newly allocated anonymous **filehandle**. Otherwise if **FILEHANDLE** is an expression, its value is the real filehandle.

File Checking

" -e " tests the existence of a file

```
if (-e "filename.cgi")
{
 #proceed with your code
}
```

To see if the file is allowed to be read, written to, or executed:

Readable: -r

Writable: -w

Executable: -x

```
$readfile="myfile.cgi";
if (-r $readfile)
{
 #proceed with your code
}
```

File Checking

Text or Binary

Text File: **-T**

Binary File: **-B**

```
$readfile="myfile.cgi";
if (-T $readfile)
{
 #proceed with your code
}
```

Multiple Tests

Test for two or more things at a time using the **"and" (&&)** or the **"or" (||)** operators.

```
$readfile="myfile.cgi";
if ( (-e $readfile) && (-r $readfile) )
{
 #proceed with your code
}
```

Perl File I/O: Examples

```
#!/usr/bin/perl
# Open file to read
open(DATA1, "<file1.txt"); # Open new file to write
open(DATA2, ">file2.txt"); # Copy data from one file to another.
while(<DATA1>) {
print DATA2 $_;
}
close( DATA1 );
close( DATA2 );
```

"$_" is a Global Special Scalar Variable, representing default input

```
[xiaoyang@orbit268l ~]$ cat file1.txt
   1 --- myshell
   2 --- argument1
   3 – argument2
[xiaoyang@orbit268l ~]$
```


What does File2.txt contain?

Perl File I/O: Examples

Lect8_6.pl

```
#!/usr/bin/perl
# Open file to read
open(INPUT, "<file1.txt");    # Open new file to write
@alllines = <INPUT>;        # Import all from a file into an array
close(INPUT);

open(OUTPUT, ">file2.txt");          # generate a file

foreach $line(@alllines){
 chomp($line);
print OUTPUT “$line”;  ###  print OUTPUT “$line\n”;
}
close( OUTPUT );
```

```
[xiaoyang@orbit268l ~]$ cat file1.txt
    My perl script
    Argument 1
    Argument 2
```

Any difference between file1.txt and file2.txt?

Open and append a file

Lect8_7.pl

```
#!/usr/bin/perl

$file1= "filename.out";
If(-e  $file1){
open(MYOUTFILE, ">>filename.out");    #open for write, append
}

print MYOUTFILE  " Steve was here\n";
print MYOUTFILE  " and now is gone\n";
print MYOUTFILE  " but left his name\n";
print MYOUTFILE  " to carry on. \n";
close(MYOUTFILE);
```

filename.out:
What message left?

What is the context in the file of "filename.out"?

Lect8_8.pl

Example

```
#!/usr/bin/perl –w
my $bigfileName = "/scratch/bigfile.txt";
my $sipfileName = "/scratch/sip.out";
my $arrayfileName = "/scratch/array.out";
my $slurpfileName = "/scratch/slurp.out";
 my $time1 = time();
print "Starting sip\n";
&sip;    #  or  sip();
print "End sip\n";
my $time2 = time();
print "Starting array\n";
buildarray();
print "End array\n";
my $time3 = time();
print "Starting slurp\n";
slurp();
print "End slurp\n";
my $time4 = time();
print "Sip time is ", $time2-$time1, " seconds\n";
print "Array time is ", $time3-$time2, " seconds\n";
print "Slurp time is ", $time4-$time3, " seconds\n";
```

```
sub sip()
{
my $inf;
my $ouf;
open $inf, “<“, $bigfileName;
open $ouf, ">“, $sipfileName;
while(<$inf>)
{
my $line = $_;
chomp $line;
print $ouf $line, "\n";
}
close $ouf;
close $inf;
}
```

Example

```
sub buildarray()
{
my $inf;
my $ouf;
my @array;
open $inf, "<" , $bigfileName;
while(<$inf>)
{
my $line = $_;
chomp $line;
push @array, ($line);
}
close $inf;
open $ouf, ">" , $arrayfileName;
foreach my $line (@array)
{
print $ouf $line, "\n";
}
close $ouf;
}
```

```
sub slurp()
{
my $inf;
my $ouf;
my $holdTerminator = $/; #new line
undef $/;
open $inf, "<" , $bigfileName;
my $buf = <$inf>;
close $inf;
$/ = $holdTerminator;
my @lines = split /$holdTerminator/, $buf;
$buf = "init";
$buf = join $holdTerminator, @lines;
open $ouf, ">" , $slurpfileName;
print $ouf $buf;
print $ouf "\n";
close $ouf;
}
```

Functions Manipulating Files

- Delete a file
- Rename a file
- Create a directory
- Remove a directory
- Managing file ownership

```
System("Linx_command files");
System("rm file");
System("mv file1 files2");
System("mkdir DIR1");
System("rm –r DIR1");
```

```
#!/usr/bin/perl –w
system("mv file1.txt file2.txt");
system("rm file1.txt);
```

Unlink ()

"**unlink()**" deletes the files specified by LIST, or the file specified by $_ otherwise. Note: no recovering once a file gets deleted!!

```
#!/usr/bin/perl –w
$file1= "/tmp/t1.txt";
If(–e $file1){
 unlink $file1;
} else{
Print "$file1 does not exist\n";
}
```

```
#!/usr/bin/perl –w
 unlink( "/tmp/t1.txt", "/tmp/t2.txt" );
```

```
#!/usr/bin/perl
my @files_to_remove = qw/file1.txt  file2.txt  file3.txt/;
my $num_removed = unlink @files_to_remove;
print "$num_removed files were removed\n";
```

Create and Remove a Directory

Lect8_9.pl

```
#!/usr/bin/perl -w

sub main {
      my $directory = "temp";
     unless(mkdir $directory)
     {
    die "Unable to create $directory \n";
       }
}

main();
```

```
#!/usr/bin/perl -w
rmdir ("/tmp/testdir") || die ("error in deleting directory: $?");
```

$?--returned by the last pipe close.

Directory Reading Routines

closedir(DIRHANDLE)
closedir DIRHANDLE

Closes a directory opened by opendir().

opendir(DIRHANDLE,EXPR)

Opens a directory named EXPR for processing by readdir(), telldir(), seekdir(), rewinddir() and closedir(). Returns true if successful.

readdir(DIRHANDLE)
readdir DIRHANDLE

Returns the next directory entry for a directory opened by opendir(). If used in an array context, returns all the rest of the entries in the directory. If there are no more entries, returns an undefined value in a scalar context or a null list in an array context.

Directory Reading Routines

rewinddir(DIRHANDLE)
rewinddir DIRHANDLE

Sets the current position to the beginning of the directory for the readdir() routine on DIRHANDLE.

seekdir(DIRHANDLE,POS)

Sets the current position for the readdir() routine on DIRHANDLE. POS must be a value returned by telldir().

telldir(DIRHANDLE)
telldir DIRHANDLE

Returns the current position of the readdir() routines on DIRHANDLE. Value may be given to seekdir() to access a particular location in a directory.

Opendir and readdir

```
# load all files of the "data/" folder into the @files array

opendir(DIR, "data/");
@files = readdir(DIR);
closedir(DIR);      # build a unsorted list from the # @files array:

print "<ul>";
foreach $file (@files) {
next if ($file eq "." or $file eq "..");
print "<li><a href=\"$file\">$file</a></li>";
}
print "</ul>";
```

rewinddir

```
#!/usr/bin/perl -w
# Open the current directory
opendir(DIR, ".");
# Print all of the directory entries.
print("1st Time: \n");
map(print("$_ \n") , readdir(DIR));
print("\n");
# Print message verifying that there are
# no more directory entries to read.
print("The last file has already been read!\n\n")
unless readdir(DIR);

# Go back to the beginning.
rewinddir(DIR);
# Print all of the directory entries again.
print("2nd Time: \n");
map(print("$_ \n") , readdir(DIR));
print("\n");
closedir(DIR);
```

map EXPR, LIST

Map( ) evaluates EXPR or BLOCK for each element of LIST. For each iteration, $_ holds the value of the current element, which can also be assigned to allow the value of the element to be updated.

telldir and seekdir

Telldir returns the current position of read pointer within the directory listing referred to by DIRHANDLE. This returned value can be used by **seekdir()** function.

```
#!/usr/bin/perl -w
opendir(DIR, "/tmp");
print("Position without read : ", telldir(DIR), "\n");
$dir = readdir(DIR);
print("Position after one read : ", telldir(DIR), "\n");
print "$dir\n";
 seekdir(DIR,0);
$dir = readdir(DIR);
print "$dir\n";
print("Position after second read : " , telldir(DIR), "\n");
 closedir(DIR);
```

When above code is executed, it produced following result

```
Position without read : 0
Position after one read : 1220443271
 test.txt
test.txt
Position after second read : 1220443271
```

glob Function

“glob” function returns a list of files **matching EXPR** as they would be expanded by the standard Bourne shell.
If the EXPR does not specify a path, uses the current directory.
If EXPR is omitted, the value of $_ is used.

```
#!/usr/bin/perl
(@file_list) = glob "perl_g*";
print "Returned list of file @file_list\n";
```

Perl Directories

```
#!/usr/bin/perl
# Display all the files in /tmp directory.
$dir = "/tmp/*";
my @files = glob( $dir );
foreach (@files )
{
print $_ . "\n";
}
```

"$_" is a Global Special Scalar Variable, representing default input

```
#!/usr/bin/perl
# Display all the hidden files.
$dir = "/tmp/.*";
@files = glob( $dir );
foreach (@files )
{
print $_ . "\n";
}
```

Perl Directories

```
#!/usr/bin/perl
# Display all the C source files in /tmp directory.
$dir = "/tmp/*.c";
@files = glob( $dir );
foreach (@files )
{
print $_ . "\n";
 }
```

```
#!/usr/bin/perl
# Display all the files from /tmp
and /home directories.
$dir = "/tmp/* /home/*";
@files = glob( $dir );
foreach (@files )
{
print $_ . "\n";
 }
```

Chdir

- **chdir EXPR**
- **chdir FILEHANDLE**
- **chdir DIRHANDLE**
- **chdir**

Changes the working directory to EXPR, if possible. If EXPR is omitted, changes to the directory specified by $ENV{HOME} , if set; if not, changes to the directory specified by $ENV{LOGDIR} .

Example

Lect8_10.pl

```
#!/usr/bin/perl
$ENVI{DATA}="/gpfs/data/xyz/GEOG760/Instructor/TEMP/";
$ENVI{SOURCE}="/gpfs/data/xyz/GEOG760/Instructor/CODE/";
chdir $ENVI{DATA};
open(DATA, ">file1.txt");
Print DATA “test the file directory\n”;
Close DATA

chdir $ENVI{SOURCE};
open(DATA, ">file2.pl");
 Print DATA “#!/usr/bin/perl \n”;
Print DATA “\$a=10; \n”;
Close DATA
If(-e “file1.txt”){
Print “file1.txt exist\n”;
}else{
Print “no such file\n”;
}
```

Which directory is file “file1.txt” located at?
Which directory is file “file2.pl” located at?
Does “file1.txt” exist?

Perl Regular Expressions

A regular **expression** is a **string** of characters that define the **pattern or patterns**.

The pattern binding operators

=~ and **!~**.

There are three regular expression operators within Perl

- Match Regular Expression **- m//**
- Substitute Regular Expression **- s///**
- Transliterate Regular Expression **- tr///**

Perl Regular Expressions

Lect8_11.pl

```
#!/usr/bin/perl
$bar = "This is food instead of foo";
if ($bar =~ m/food/){
print "First time is matching\n";
}else{
print "First time is not matching\n";
 }
```

s/PATTERN/REPLACEMENT/;

```
#/usr/bin/perl
$string = "The cat sat on the mat";
$string =~ s/cat/dog/;
print "$string\n";
```

Transliterate Regular Expression

tr/SEARCHLIST/REPLACEMENTLIST/

```
#/user/bin/perl
$string = 'The cat sat on the mat';
$string =~ tr/a/o/;
print "$string\n";

##This will produce following result
The cot sot on the mot.
```

Perl Regular Expressions

Match Characters

char	meaning
^	beginning of string
$	end of string
.	any character except newline
*	match 0 or more times
+	match 1 or more times
?	match 0 or 1 times; *or*: shortest match
|	alternative
( )	grouping; "storing"
[ ]	set of characters
{ }	repetition modifier
\	quote or special

/^Rob/ matches "Robert", "Rob"

/txt$/ matches "a.txt", Not "a.txtfile"

/.ob/ matches "bob', "robert","knob"

/log[0-9]*/ matches "log", "log1", "log999"

/log[0-9]+/ matches "log1", "log999", Not "log"
/html?/ matches "html", "htm", "ht"

/mar.*|ash.*/ matches patterns all mar or ash

^ --circumflex

Examples

Lect8_13.pl

```
#!/usr/bin/perl
$mystring = “Date in day of year is 2016365”;
If($mystring=~m/^day/){
Print “find the string\n”;
}else{
Print “it’s a wrong string\n”;
}
If($mystring=~m/[0-9]$/){
Print “find the string\n”;
}

If($mystring=~m/day/){
Print “find the string\n”;
}else{
Print “it’s a wrong string\n”;
}
```

Examples

Lect8_14.pl

```
#!/usr/bin/perl
$mystring = “Date in day of year is 2016365”;
If($mystring=~m/^Date/){
Print “find the string\n”;
}else{
Print “it’s a wrong string\n”;
}

If($mystring=~m/(\d+)$/){
Print “find the string\n”;
}else{
Print “it’s a wrong string\n”;
}
```

Examples

Lect8_15.pl

```
#!/usr/bin/perl
$mystring = “Date in day of year is 2016365″;
If($mystring=~m/[0-9]/){
Print “find the string\n”;
}else{
Print “it’s a wrong string\n”;
}

If($mystring=~m/2016$/){
Print “find the string\n”;
}else{
Print “it’s a wrong string\n”;
}
```

Pattern: Characters with Special Meaning

Characters that have special meaning in a regular expression can be included literally by preceding them with an escape character (\).

Lect8_16.pl

```
#!/usr/bin/perl
@flist=(
“drwx------  32 zhangx zhangx 12288 Sep 17 11:21”,
“drwxr-xr-x.  9 root   root    4096 Jun 11 16:49”,
“drwxrwxr-x   2 zhangx  zhangx  4096 Sep 13  2013 test1.pl”,
“-rw-------   1 zhangx zhangx 27106 Sep 17 08:23 .bash_history”,
);
foreach $line(@flist){
if($line=~/\.pl$/){
($x, $y)=(split(‘ ’, $line))[0, 7];
}
print $x, $y, “\n”;
}
```

Character Sets: Specialties Inside

Character sets: specialities inside [...]

Different meanings apply inside a character set ("character class") denoted by [...] so that, **instead of** the normal rules given here, the following apply:

[*characters*]	matches any of the characters in the sequence
[*x*-*y*]	matches any of the characters from *x* to *y* (inclusively) in the ASCII code
[\-]	matches the hyphen character "-"
[\n]	matches the newline; other single character denotations with \ apply normally, too
[^*something*]	matches any character *except* those that [*something*] denotes; that is, immediately after the leading "[", the circumflex "^" means "not" applied to all of the rest

Perl Regular Expressions

\d - a digit - - [0-9]
\D - a nondigit -- [^0-9]
\w - a word character (alphanumeric including underscore) -- [a-zA-Z_0-9]
\W - a nonword character -- [^a-zA-Z_0-9]
\s - a whitespace character -- [\t\n\r\f]
\S - a non-whitespace character -- [^ \t\n\r\f]

- **\B** - Matches everywhere except between a word character and non-word character
- **\b** - Matches between word character and non-word character
- **\A** - Matches only at the beginning of a string
- **\Z** - Matches only at the end of a string or before a newline
- **\z** - Matches only at the end of a string
- **\G** - Matches where previous m//g left of

Perl Regular Expressions: Examples

Examples

expression	matches...
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c

Perl Regular Expressions: Examples

Examples

expression	matches...
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}
\w+	a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1
100\s*mk	the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)
abc\b	abc when followed by a word boundary (e.g. in abc! but not in abcd)
perl\B	perl when *not* followed by a word boundary (e.g. in perlert but not in perl stuff)

Example

Lect8_17.pl

```
#!/usr/bin/perl
$mystring = “Today is [2019/10/14].”;
@myarray = ($mystring =~ m/(\d+)/g);
print join(“_", @myarray);
```

g – global search

Print out:

2019_10_14.

Example

Lect8_20.pl

```
#!/usr/bin/perl
$mystring = “Date in day of year is 2014365.";
If($mystring=~m/([0-9]{4})([0-9]{3})/){
$year=$1;
$doy=$2;
}


Print “year is $year\n”;
Print “year is $doy\n”;
```

Example

Lect8_21.pl

```
#!/usr/bin/perl
@flist=(
“drwxrwxr-x  zhangx 4096 Sep-13 10:20 Data/TEMP/test1.pl”,
“-rw-------  zhangx 27106 Sep-17 08:23 .bash_history.pl”
);
Foreach $line(@flist){
If($line=~m!([-drwxs]{10})\s+.*\s+.+/(.*pl)$!){
Print “filename: $2\n”;
Print “perl string: $1\n”;
}
}
```

filename: test1.pl
perl string: drwxrwxr-x

drwxrwxr-x zhangx 4096 Sep-13 10:20 Data/test1.pl

m!([-drwxs]{10})\s+.*\s+.+/ (.*pl)$!

Example

Lect8_22.pl

```
#!/usr/bin/perl
@flist=(
“-rw-rw-r--  zhangx 4096 2014-10-14 10:20 test1.pl”,
“-rw------- zhangx 27106 2013-09-17 08:23 history.pl”,
);
Foreach $line(@flist){
If($line=~m!^([-rwxstl]{10})\s+.*(\d{4}-\d{2}-\d{2})\s+.*\s+(.*pl)$ !){
Print “$1\n”;
Print “$2\n”;
Print “$3\n”;
}
}
```

```
-rw-rw-r--
2014-10-14
test1.pl
-rw-------
2013-09-17
history.pl
```

Notice

Perl is a language that's designed for text processing and is an **interpreted** programming language. For data computing, the languages such as C/C++, FORTRAN, IDL, should be used.

Summary

Subroutines and Parameters
Pattern Matching
String Manipulation
File and Directory I/O
Input/Output Processing

Midterm Exam

20 multiple choices:

To kill a background process with ID 51 using the following command:
kill -1 51
kill 51%
kill %51
kill -9 51
kill -9 51%

Find error from four pieces of codes:

```
#!/bin/sh
# My first shell script
echo "I love computer game";
x=10;
y=20;
z=x+y
echo "value for x, y and z is" $x, $y, z
```