Week 09 ▾ Tutorial ▾

COMP[29]041 18s2

Sample Answers ▾

1. How is the assignment going?
   Does anyone have hints or advice for other students?

   Has anyone discovered interesting cases that have to be handled?

   Discussed in tutorial.

2. You work on the assignment for an hour tonight. What do you need to do when you are finished?

   1. Update your **diary.txt** with a line indicating you work for an hour and give a brief breakdown of what the work was: (was it coding, debugging, testing ...)
   2. submit the latest version of your code with give. Do this every time you work on the assignment.

3. Apart from **legit.pl** and **diary.txt** what else do you need to submit (and give an example)?

   10 test files - **test00.sh** .. **test09.sh**
   Here is an example of a suitable test file:

```
#!/bin/sh
# check that add works combined with commit -a
legit.pl init
echo line 1 >a
legit.pl add a
legit.pl commit -m 'first commit'
echo line 2 >>a
echo world >b
legit.pl add b
legit.pl commit -a -m 'second commit'
legit.pl show 1:a
legit.pl show 1:b
```

4. What is a merge conflict - and how do they get handled in git and legit?

   A merge conflict occurs when we attempt to merge branches and conflicting changes to the same part of a file have occured in both branches.
   legit just stops with an error.
   git shows you the conflicting changes and lets you resolve the conflict.

5. What is bash arithmetic? What are its advantages and disadvantages.

   Arithmetic was originally provide in shell scripts by external programs **test** and **expr**.
   For example:

```
number=1
while test $number -le 42
do
    echo $number
    number=`expr $number + 1`
done
```

   Bash added an arithmetic syntax and most other commonly-used eventually added the same syntax.

```
number=1
while ((number <= 42))
do
    echo $number
    number=$((number + 1))
done
```

   Bash arithmetic is more readable and faster but you may encountered shell, e.g. on embedded devices which don't support bash arithmetic.

6. Discuss what this Shell script does then translate it directly to Perl using Perl's **system** fucntion.
   Then translate it to Perl than doesn't use **system**.

```
#!/bin/sh

important_file=/home/cs2041/public_html/index.html

while ls $important_file >/dev/null 2>&1
do
        echo "all OK"
        sleep 1
done

echo "Panic $important_file gone"
```

The shell script every second checks to see if a particular file exists.
If it doesn't the script exists prints a message and exits.

Here is a translation using **system**.

Note an exit code of 0 from ls indicates success. But 0 is false in Perl (and C and Python and ...)

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

$important_file = "/home/cs2041/public_html/index.html";

while (!system "ls $important_file >/dev/null 2>&1") {
        system "echo \"all OK\"";
        system "sleep 1";
}
system "echo \"Panic $important_file gone\"";
```

Here is more natural Perl code using the equivalent Perl buitlin functions.questions

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

$important_file = "/home/cs2041/public_html/index.html";

while (-e $important_file) {
        print "all OK\n";
        sleep 1;
}
print "Panic $important_file gone\n";
```

7.
```
#!/bin/bash
number=0
while test $number -ge 0
do
    echo -n "Enter number: "
    read number
    if  test $number -ge 0
    then
        if  test `expr $number % 2` -eq 0
        then
            echo "$number is even"
        else
            echo "$number is odd"
        fi
    fi
done
echo "Bye"
```

Discuss what has to be done to transform the above Shell using test/expr to bash arithmetic

- change test to (( ))
- change expr to $(( ))
- remove $ from variables in both

8.
```
#!/bin/bash
number=0
while ((number >= 0))
do
    echo -n "Enter number: "
    read number
    if ((number >= 0))
    then
        if ((number % 2 == 0))
        then
            echo "$number is even"
        else
            echo "$number is odd"
        fi
    fi
done
echo "Bye"
```

Discuss what has to be done to transform the above shell to equivalent Perl.

- change #! line
- add semi-colons
- put $ in front of variables
- add braces ({}) where indenting indicates
- chang double brackets to single bracket on if & while statments
- add chomp
- change echo to print and add \n

```
#!/usr/bin/perl -w
$number = 0;
while ($number >= 0) {
    print "Enter number: ";
    $number = <STDIN>;
    chomp $number;
    if ($number >= 0) {
        if ($number % 2 == 0) {
            print "$number is even\n";
        } else {
            print "$number is odd\n";
        }
    }
}
print "Bye\n";
```

Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session.
Your tutor may still choose to cover some of the questions time permitting.

9. Write a Perl program, `tags.pl` which given the URL of a web page fetches it by running *wget* and prints the HTML tags it uses. The tag should be converted to lower case and printed in sorted order with a count of how often each is used.

   Don't count closing tags.

   Make sure you don't print tags within HTML comments.

   For example:

```
$ ./tags.pl http://www.cse.unsw.edu.au
a 141
body 1
br 14
div 161
em 3
footer 1
form 1
h2 2
h4 3
h5 3
head 1
header 1
hr 3
html 1
img 12
input 5
li 99
link 3
meta 4
noscript 1
p 18
script 14
small 3
span 3
strong 4
title 1
ul 25
```

Note the counts in the above example will not be current - the CSE pages change almost daily.

Sample solution for tags.pl

```
#!/usr/bin/perl -w
# written by andrewt@cse.unsw.edu.au as a COMP2041 example
# fetch specified web page and count the HTML tags in them

# There are better ways to fetch web pages (e.g. HTTP::Request::Common)
# The regex code below doesn't handle a number of cases.  It is often
# better to use a library to properly parse HTML before processing it.
# But beware illegal HTML is common & often causes problems for parsers.

foreach $url (@ARGV) {
    $webpage = `wget -q -O- '$url'`;
    $webpage =~ tr/A-Z/a-z/;
    $webpage =~ s/<!--.*?-->//g; # remove comments
    @tags = $webpage =~ /<\s*(\w+)/g;
    foreach $tag (@tags) {
        $tag_count{$tag}++;
    }
}
foreach $tag (sort keys %tag_count) {
    print "$tag $tag_count{$tag}\n";
}
```

10. Add an -f option to tags.pl which indicates the tags are to printed in order of frequency.

```
$ tags.pl -f http://www.cse.unsw.edu.au
head 1
noscript 1
html 1
form 1
title 1
footer 1
header 1
body 1
h2 2
hr 3
h4 3
span 3
link 3
small 3
h5 3
em 3
meta 4
strong 4
input 5
img 12
br 14
script 14
p 18
ul 25
li 99
a 141
div 161
```

Sample solution for tags.pl

```
#!/usr/bin/perl -w
# written by andrewt@cse.unsw.edu.au as a COMP2041 example
# fetch specified web page and count the HTML tags in them

# The regex code below doesn't handle a number of cases.  It is often
# better to use a library to properly parse HTML before processing it.
# But beware illegal HTML is common & often causes problems for parsers.

use LWP::Simple;

$sort_by_frequency = 0;
foreach $arg (@ARGV) {
    if ($arg eq "-f") {
        $sort_by_frequency = 1;
    } else {
        push @urls, $arg;
    }
}
foreach $url (@urls) {
    $webpage = get $url;
    $webpage =~ tr/A-Z/a-z/;
    $webpage =~ s/<!--.*?-->//g; # remove comments
    @tags = $webpage =~ /<\s*(\w+)/g;
    foreach $tag (@tags) {
        $tag_count{$tag}++;
    }
}
if ($sort_by_frequency) {
    @sorted_tags = sort {$tag_count{$a} <=> $tag_count{$b} || $a cmp $b} keys %tag_count;
} else {
    @sorted_tags = sort keys %tag_count;
}
print "$_ $tag_count{$_}\n" foreach @sorted_tags;
```

COMP[29]041 18s2: Software Construction is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au