Week 09 Laboratory Sample Solutions

COMP[29]041 18s2

Objectives

- Practicising Git, Perl & Shell

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab09` by typing:

```
$ mkdir lab09
```

Change to this directory by typing:

```
$ cd lab09
```

Exercise: Resolving A Merge Conflict

This lab exercise will involve solving a git merge conflict.
Two friends have decided to revise together by re-implementing a lab exercise **tail.pl** - a Perl program which prints the last n lines of a file.

They start out together at CSE:

```
$ mkdir tail
$ cd tail
$ git init
Initialized empty Git repository in tail/.git/
$ gedit tail.pl &
$ git add tail.pl
$ git commit -a -m "first commit"
[...]
```

Then they go home and work separately. One friend creates a new branch and works in it.

```
$ git checkout -b friend
Switched to a new branch 'friend'
$ gedit tail.pl &
$ git commit -a -m "My fantastic attempt at tail.pl"
[...]
```

The other keeps working in the master branch like this:

```
$ git checkout master
Switched to branch 'master'
$ gedit tail.pl &
$ git commit -a -m "my attempt"
[...]
```

You job is to help the two friends out by merging the two branches. Download tail.zip here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/merging_tails/tail.zip .
```

Unfortunately merging them together you get a merge conflict, as **tail.pl** has conflicting updates committed to it, in both of the branches.

```
$ unzip tail.zip
....
$ cd tail
$ ls
tail.pl
$ git status
On branch master
nothing to commit, working tree clean
$ git branch
  friend
* master
$ git merge friend
Auto-merging tail.pl
CONFLICT (content): Merge conflict in tail.pl
Automatic merge failed; fix conflicts and then commit the result.
```

Open the resulting tail.pl file in an editor of your choice and you will notice that git has spliced the two files together and made some decisions about where the files differ, in the following Format:

```
[... code that seems the same between the files ...]
<<<<<<<< HEAD
[... code from the branch you are currently on ...]
=======
[... code from the branch you tried to merge ...]
>>>>>>>> BRANCH
[... more code that git has decided is the same ...]
```

Both of the target tail.pl files contain correct and incorrect code, and it is up to you to decide what parts to keep or to remove.

The code of both friends is confusing so pay careful attention to syntax, variable choices, and structure.

Edit and clean up the resulting tail.pl, also removing any of the characters inserted during the git merge, so that it will work as intended.

When finished, perform a final commit and submit a zip of the whole git repo.

```
$ gedit tail.pl &
$ git commit -a -m merged
....
$ git gc
$ zip -r git.zip .git
```

git gc cleans up and compresses git's internal files

You should have finished up with Perl similar to the code below, but this exercise was really about getting you used to resolving merge conflicts , so ask your tutor if there any part of the merge process you don't understand.

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

$max_lines = 10;

if (@ARGV > 0 && $ARGV[0] =~ /-([0-9]+)/) {
    $max_lines = $1;
    shift @ARGV;
}

if (@ARGV == 0) {
    @lines = <>;
    $first = @lines - $max_lines;
    $first = 0 if $first < 0;
    print @lines[$first..$#lines];
} else {
    $show_file_names = @ARGV > 1;
    foreach $file (@ARGV) {
        open my $f, '<', $file or die "$0: can't open $file\n";
        print "==> $file <==\n" if $show_file_names;
        @lines = <$f>;
        $first = @lines - $max_lines;
        $first = 0 if $first < 0;
        print @lines[$first..$#lines];
        close $f;
    }
}
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest merging_tails
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab09_merging_tails git.zip
```

Exercise: Translating Bash to Perl

We have some Shell (Bash) scripts which do arithmetic calculations which we need to translate to Perl.
Write a Perl program **shell_translator.pl** which takes such a Bash script as input and outputs an equivalent Perl program.

The scripts use the arithmetic syntax supported by Bash (and several other shells). Fortunately the scripts only use a very limited set of shell features.

You can assume all the features you need to translate are present in the following 4 examples.

- sum.sh sums a series of integers:

```
$ cat sum.sh
#!/bin/bash

# sum the integers $start .. $finish

start=1
finish=100
sum=0

i=1
while ((i <= finish))
do
    sum=$((sum + i))
    i=$((i + 1))
done

echo Sum of the integers $start .. $finish = $sum
$ ./sum.sh
Sum of the integers 1 .. 100 = 5050
$ ./shell_translator.pl sum.sh
#!/usr/bin/perl -w
# sum the integers $start .. $finish
$start = 1;
$finish = 100;
$sum = 0;
$i = 1;
```

Or you can put the Perl in a file and run it:

```
$ ./shell_translator.pl sum.sh >sum.pl
$ chmod 700 ./sum.pl
$ ./sum.pl
Sum of the integers 1 .. 100 = 5050
```

- double.sh prints powers of two:

```
$ cat double.sh
#!/bin/bash

# calculate powers of 2 by repeated addition

i=1
j=1
while ((i < 31))
do
    j=$((j + j))
    i=$((i + 1))
    echo $i $j
done
$ ./shell_translator.pl double.sh
#!/usr/bin/perl -w
# calculate powers of 2 by repeated addition
$i = 1;
$j = 1;
while ($i < 31) {
    $j = $j + $j;
    $i = $i + 1;
    print "$i $j\n";
}
$ ./shell_translator.pl double.sh|perl
2 2
3 4
```

- pythagorean_triple.sh searches for Pythagorean triples:

```
$ cat pythagorean_triple.sh
#!/bin/bash

max=42
a=1
while ((a < max))
do
    b=$a
    while ((b < max))
    do
        c=$b
        while ((c < max))
        do
            if ((a * a + b * b == c * c))
            then
                echo $a $b $c is a Pythagorean Triple
            fi
            c=$((c + 1))
        done
        b=$((b + 1))
    done
    a=$((a + 1))
done
$ ./shell_translator.pl pythagorean_triple.sh
#!/usr/bin/perl -w
```

- collatz.sh prints an interetsing series:

```
$ cat collatz.sh
#!/bin/bash

# https://en.wikipedia.org/wiki/Collatz_conjecture
# https://xkcd.com/710/

n=65535
while ((n != 1))
do
    if ((n % 2 == 0))
    then
        n=$((n / 2))
    else
        n=$((3 * n + 1))
    fi
    echo $n
done
$ ./shell_translator.pl collatz.sh
#!/usr/bin/perl -w
# https://en.wikipedia.org/wiki/Collatz_conjecture
# https://xkcd.com/710/
$n = 65535;
while ($n != 1) {
    if ($n % 2 == 0) {

        $n = $n / 2;
```

Note: you must translate the scripts to Perl not just embed the shell of the script in a Perl system call.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest shell_translator
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab09_shell_translator shell_translator.pl
```

Sample solution for `shell_translator.pl`

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

# Translate some Bash statements to perl
# written by andrewt@unsw.edu.au as COMP[29]041 sample solution

sub translate_line {
    my ($line) = @_;

    # remove trailing white space
    $line =~ s/\s*$//;

    # remove and save leading white space
    $line =~ s/^(\s*)//;
    my ($indent) = $1;

    # remove and save any comment
    $line =~ s/(\s*#.*)//;
    my ($comment) = $1;

    my $perl = translate_bash($line);

    if ("$perl$comment" ne "") {
        return "$indent$perl$comment\n";
    } else {
        return "";
    }
}

sub translate_bash {
    my ($bash) = @_;

    my $perl = "";
    if ($bash =~ /^(do|then)$/) {
        $perl = "";
    } elsif ($bash =~ /^(done|fi)$/) {
        $perl = "}";
    } elsif ($bash =~ /^else$/) {
        $perl = "} else {";
    } elsif ($bash =~ /^echo\s+(.*)$/) {
        $perl = "print \"$1\\n\";";
    } elsif ($bash =~ /^(if|while)\s+\(\((.*)\)\)$/) {
        my $keyword = $1;
        my $expression = $2;
        # add $ in front of any variable that don't already have a $
        $expression =~ s/(^|[^\$])([a-zA-Z]\w*)/$1\$$2/g;
        $perl = "$keyword ($expression) {";
    } elsif ($bash =~ /(.*?)=\$\(\((.*)\)\)/) {
        my $lhs = $1;
        my $expression = $2;
        $expression =~ s/(^|[^\$])([a-zA-Z]\w*)/$1\$$2/g;
        $perl = "\$$lhs = $expression;";
    } elsif ($bash =~ /(.*?)=(.*)/) {
        my $lhs = $1;
        my $rhs = $2;
        $rhs =~ s/(^|[^\$])([a-zA-Z]\w*)/$1\$$2/g;
        $perl = "\$$1 = $2;";
    }

    return $perl;
}

while ($line = <>) {
    if ($. == 1 && $line =~ /^#!/) {
        print "#!/usr/bin/perl -w\n";
    } else {
        print translate_line($line);
    }
}
```

Alternative solution for `shell_translator.pl`

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

# translate some bash statements to perl
# written by andrewt@unsw.edu.au as COMP[29]041 sample solution
#
# Very terse Perl solution using $_ and Perl's -p command line option.
#
# Good example of how you can use Perl to produce code quickly
# at the cost of readability and maintainability.
# Good approach for one-off/throwaway code where you check correctness manually


s?^#!.*?#!/usr/bin/perl -w? && next if $. == 1;
/^\s*#/ && next;
s/^(\s*)(do|then)\b.*\n// && next;
s/^(\s*)(done|fi)\b/$1}/ && next;
s/^(\s*)(else)\b/$1} else {/ && next;
s/^(\s*)echo\s+(.*)/$1print "$2\\n";/ && next;
s/\$\(\((.*)\)\)/$1/;
s/(^|[^\$])([a-zA-Z]\w*)/$1\$$2/g;
s/^(\s*)\$(if|while)\s+\(\((.*)\)\)/$1$2 ($3) {/ && next;
s/(.*\w)=(.*)/$1 = $2;/;
```

Challenge Exercise: A Perl Program that Prints Perl

Write a Perl program **perl_print.pl** which is given a single argument. It should output a Perl program which when run, prints this string. For example:

```
$ ./perl_print.pl 'Perl that prints Perl - yay' |perl
Perl that prints Perl - yay
```

You can assume the string contains only ASCII characters. You can not make other assumptions about the characters in the string.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_print
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab09_perl_print perl_print.pl
```

Sample solution for `perl_print.pl`

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

# Output a Perl program which when run will print the
# string supplied as a commandline argument
# written by andrewt@unsw.edu.au as COMP[29]041 sample solution

print "print \"";

foreach $argument (@ARGV) {
    for $c (split //, $argument) {
        # translate everything but word characters to a hexadecimal escape
        if ($c =~ /\w/) {
            print $c;
        } else {
            printf "\\x%02x", ord($c);
        }
    }
}

print "\\n\";\n";
```

Challenge Exercise: A Perl Program that Prints Perl that Prints Perl that ...

Write a Perl program `perl_print_n.pl` which is given a two arguments, an integer **n** and a string.
If **n** is 1 it should output a Perl program which prints a Perl program which prints a Perl program which prints the string.

If **n** is 2 it should output a Perl program which prints a Perl program which prints a Perl program which prints a Perl program which prints a Perl program which prints the string.

And so on for larger values of **n**.

For example:

```
$ ./perl_print_n.pl 1 'Perl that prints Perl'
print "Perl that prints Perl\n"
$ ./perl_print_n.pl 2 'Perl that prints Perl that Prints Perl'|perl|perl
Perl that prints Perl that Prints Perl
$ ./perl_print_n.pl 2 'Perl that ....'|perl|perl
Perl that ....
$ ./perl_print_n.pl 4 'Andrew Rocks!'|perl|perl|perl|perl
Andrew Rocks!
$ ./perl_print_n.pl 10 'I love COMP[29]041!'|perl|perl|perl|perl|perl|perl|perl|perl|perl|perl
I love COMP[29]041!
```

You can assume the string contains only ASCII characters. You can not make other assumptions about the characters in the string.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_print_n
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab09_perl_print_n perl_print_n.pl
```

Sample solution for `perl_print_n.pl`

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

sub print_n {
    my ($n, $string) = @_;

    for (1 .. $n) {
        my $escaped_chars = escape_string($string);
        $string = "print \"$escaped_chars\\n\"";
    }

    return $string;
}

sub escape_string {
    my ($string) = @_;
    my $escaped_chars = "";
    for $c (split //, $string) {
        # Converts double-quotes, backslash and non-printing characters except space & tab
        # into hex escape sequences (if we escape all characters generated program size explodes)
        if ($c =~ /[^[:print:] \t]/ || $c eq '"' || $c eq '\\') {
            $escaped_chars .= sprintf "\\x%02x", ord($c);
        } else {
            $escaped_chars .= $c;
        }
    }
    return $escaped_chars;
}

die "Usage: $0 <n> <string\n" if @ARGV != 2;
print print_n($ARGV[0], $ARGV[1]), "\n";
```

Submission

When you are finished each exercises make sure you submit your work by running **give**.
You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Saturday 5 January 19:00** to submit your work.

You cannot obtain marks by e-mailing lab work to tutors or lecturers.

You check the files you have submitted here

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The lab exercises for each week are worth in total 1 mark.
The best 10 of your 11 lab marks will be summed to give you a mark out of 9. If their sum exceeds 9 - your mark will be capped at 9.

- You can obtain full marks for the labs without completing challenge exercises
- You can miss 1 lab without affecting your mark.

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
CRICOS Provider 00098G