Week 05 Tutorial Sample Answers

1. The assignment specification doesn't fully explain the assignment - what can I do?

   A big part of the assignment is understanding what git does.
   You'll spend more time doing this than implementing subset 0.

   On line tutorials can help with this.

   You can also use the the reference implementation **2041 shrug** to discover what your program is supposed to do in any situation.

2. How hard are the subsets?

   Once you understand what you have to dom subset 0 is not that hard.
   Subset 1 is hard.

   Subset 2 really hard.

   But note the marking scheme recognizes the difficulty of subsets 1 & 2.

3. What is `gitlab.cse.unsw.edu.au` and how do I want use it for ass1?

   Its a server run by CSE to host git repositories of student programs for some CSE courses.
   As assignment 1 is to implement a version control system similar to git, it is too confusing to ask you to use git for the assignment.

   Instead **give** will maintain a git repository for the assignment for you.

   Every you run give for the assignment, it will create a commit in a git repository and it will push the commit to https://gitlab.cse.unsw.edu.au/ when you can access it.

   Everytime you work on the assignment you should run give when you finish.

   This ensures you have a complete backup of all work on your program and can return to its state at any stage.

   It will also allow your tutor to check you are progressing on the lab as they can access your gitlab repository

4. What does **git init** do?

   How does this differ from **shrug-init**

   git init creates an empty repository as does **shrug-init**

   git init uses the sub-directory **.git** (by default)

   shrug-init uses the sub-directory **.shrug** inside **.git**

   git init creates many files and subdirectories

   shrug-init only ÕĠĠǨÝ to create **.shrug** but ùl Õ create other things inside **.shrug**

5. What do **git add *file*** and **shrug-add *file*** do?

   Adds a copy of ***file*** to the repository's **index**.

6. What is the index in **shrug** (and **git**), and where does it get stored?

   Files get added to the repositoy via the index so its somethimes called a staging area.
   It must be stored in the **.shrug** directory. exactly how you store it is up to you.

   You might create a directory **.shrug/index/** and store the files there.

7. What is a commit in **shrug** (and **git**), and where does it get stored?

   A commit preserves the state of all files in the index.
   It must be stored in **.shrug**. exactly how you store it is up to you.

   You might create a directory **.shrug/*commit-number*/** and store the files there.

8. Discuss what **./shrug-status** should print below?

```
$ ./shrug-init
Initialized empty shrug repository in .shrug
$ touch a b c d e f g h
$ ./shrug-add a b c d e f
$ ./shrug-commit -m 'first commit'
Commited as commit 0
$ echo hello >a
$ echo hello >b
$ echo hello >c
$ ./shrug-add a b
$ echo world >a
$ rm d
$ ./shrug-rm e
$ ./shrug-add g
$ ./shrug-status
```

```
$ ./shrug-status
a - file changed, different changes staged for commit
b - file changed, changes staged for commit
c - file changed, changes not staged for commit
d - file deleted
e - deleted
f - same as repo
g - added to index
h - untracked
# plus all shrug-* files will be untracked
```

9. Apart from the **shrug-*** and **diary.txt** scripts 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/dash

   # check that add works combined with commit -a
   shrug-init
   echo line 1 >a
   shrug-add a
   shrug-commit -m 'first commit'
   echo line 2 >>a
   echo world >b
   shrug-add b
   shrug-commit -a -m 'second commit'
   shrug-show 1:a
   shrug-show 1:b
   ```

10. 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: (coding, debugging, testing ...)
    2. submit the latest version of your code with give. Do this every time you work on the assignment.

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

    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.
    shrug just stops with an error.

    git shows you the conflicting changes and lets you resolve the conflict.

12. Write a Perl function which takes an integer argument **n** and reads the next **n** lines of input and returns them as a string.

    Two sample solutions with extra code to run the function:

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

$n = shift @ARGV or die "Usage: $0 <n-lines>\n";

sub n_lines0 {
    my ($n) = @_;
    my $text = "";
    while ($n-- > 0) {
        $text .= <>;
    }
    return $text;
}

sub n_lines1 {
    my ($n) = @_;
    my $text = "";
    $text .= <> foreach (1..$n);
    return $text;
}

print n_lines1($n);
```

13. Write a Perl program which given the name of a C function searches the C source files (*.c) in the current directory for calls of the function, declarations & definitons of the function and prints a summary indicating the file and line number, in the format below.
You can assume functions are defined with the type, name and paramaters on a single non-indented line. You can assume function bodies are always indented.

You don't have to handle multi line comments. Try to avoid matching the function name in strings or single line comments. For example:

```
$ cat half.c
double half(double x) {
    return  x/2;
}
$ cat main.c
#include <stdio.h>
#include <stdlib.h>

double half(double x);

int main(int argc, char *argv[]) {
    return  half(atoi(argv[1]));
}
$ ./print_function_uses.pl half
a.c:1 function half defined
half.c:1 function half defined
main.c:4 function half declared
main.c:7 function half used
```

Perl sample solution

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

$function = $ARGV[0] or die "Usage: $0 <function-name>\n";

foreach $c_file (glob "*.c") {
    open my $cf, '<', $c_file or die "$0: can not open $c_file: $!\n";
    while ($line = <$cf>) {
        # remove single-line comments & strings (breaks if strings contain ")
        $line =~ s/\/\/.*//;
        $line =~ s/\/\*.*?\*\///;
        $line =~ s/".*?"//;
        # note use of \b (word boundary) to match function
        $line =~ /\b$function\s*\(/ or next;
        print "$c_file:$. function $function ";
        # if line is indented it should be a call to the function
        if ($line =~ /^\s/) {
            print "used\n";
        } elsif ($line =~ /;/) {
            print "declared\n";
        } else {
            print "defined\n";
        }
    }
    close $cf;
}
```

git checkout $w sample solution

```
#!/usr/bin/python
import glob, sys, re

if len(sys.argv) != 2:
    sys.stdout.write("Usage: %s <function-name>\n\n" % sys.argv[0])
    sys.exit(1)

function = sys.argv[1]

for c_file in glob.glob("*.c"):
    with open(c_file) as cf:
        # note use of \b (word boundary) to match function
        function_regex = r'\b%s\s*\(' % function
        line_number = 0
        for line in cf:
            line_number = line_number + 1
            # remove single-line comments & strings (breaks if strings contain ")
            line = re.sub(r'\/\/.*', '', line)
            line = re.sub(r'".*?"', '', line)
            if not re.search(function_regex, line):
                continue
            # if line is indented it should be a call to the function
            if re.search(r'^\s', line):
                which = "used"
            elif re.search(r';', line):
                which = "declared"
            else:
                which = "defined"
            print("%s:%d function %s %s" % (c_file, line_number, function,  which))
```

14. Write a Perl program which given a C program as input finds the definitions of single parameter functions and prints separately the function's type, name and the parameters name & type. Assume all these occur on a single non-indented line in the C source code. You can assume function bodies are always indented. Allow for white space occuring anywhere in the function header. You can assume that types in the program don't contain square or round brackets. For example:

```
$ cat a.c
double half(int *x) {
    return  *x/2.0;
}
$ ./print_function_types.pl a.c
function type='double'
function name='half'
parameter type='int *'
parameter name='x'
```

Perl sample solution

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

while ($line = <>) {
    $line =~ /^([a-zA-Z_].*)\((.*)\)/ or next;
    $function_start = $1;
    $parameter = $2;
    $function_type = $function_start;
    $function_type =~ s/\s*([a-zA-Z_]\w*)\s*$// or next;
    $function_name = $1;
    $parameter_type = $parameter;
    $parameter_type =~ s/\s*([a-zA-Z_]\w*)\s*$// or next;
    $parameter_name = $1;
    print "function type='$function_type'\n";
    print "function name='$function_name'\n";
    print "parameter type='$parameter_type'\n";
    print "parameter name='$parameter_name'\n";
}
```

15. Write a Perl script C_include.pl which given the name of a C source file prints the file replacing any '#include' lines with the contents of the included file, if the included file itself contains a #include line these should also be processed.
    Assume the source files contain only quoted ("") include directives which contain the files's actual path name. For example:

```
$ cat f.c
#include "true.h"

int main(int argc, char *argv[]) {
    return  TRUE || FALSE;
}
$ cat true.h
#define TRUE 1
#include "false.h"
$ cat false.h
#define FALSE 0
$ ./C_include.pl f.c
#define TRUE 1
#define FALSE 0

int main(int argc, char *argv[]) {
    return  TRUE || FALSE;
}
```

Perl sample solution

```
#!/usr/bin/perl -w
# Given C source files interpolate #include "FILE" directives recursively.
sub include_file($);

sub include_file($) {
    my ($file) = @_;
    # this function is recursive so a local filehandle is essential
    open my $f, '<', $file or die "$0: can not open $file: $!";
    while ($line = <$f>) {
        if ($line =~ /^#\s*include\s*"([^"]*)"/) {
            include_file($1);
        } else {
            print $line;
        }
    }
    close $f;
}

foreach $file (@ARGV) {
    include_file($file);
}
```

16. Modify C_include.pl so that it handles both "" and <> directives. It should search the directories `/usr/include/`, `/usr/local/include` and `/usr/include/x86_64-linux-gnu` for include files specified in <> directives and for files specified in "" directives which do not exist locally. For example:

```
$ cat g.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("hello world\n");
    exit(0);
}
$ ./C_include.pl g.c|head
./C_include.pl: can not find: bits/libc-header-start.h
/* Define ISO C stdio on top of C++ iostreams.
   Copyright (C) 1991-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
```

Perl sample solution

```perl
#!/usr/bin/perl -w
# Given C source files interpolate #include "FILE" and #include <FILE>
# directives recursively.
# The recursion in this script may not terminate on stdio.h etc
# because #ifdef directive are not handled

@include_dirs = ('/usr/include/','/usr/local/include','/usr/include/x86_64-linux-gnu');

sub include_file($@);

sub include_file($@) {
    my ($file,@prefixes) = @_;
    foreach $prefix (@prefixes) {
        # this function is recursive so a local filehandle is essential
        my $path = "$prefix$file";
        next if !-r $path;
        open my $f, '<', $path or die "$0: can not open $path: $!";
        while ($line = <$f>) {
            if ($line =~ /^#\s*include\s*"(.*?)"/) {
                include_file($1, ('',@include_dirs));
            } elsif ($line =~ /^#\s*include\s*<(.*?)>/) {
                include_file($1, @include_dirs);
            } else {
                print $line;
            }
        }
        close $f;
        return;
    }
    die "$0: can not find: $file\n";
}

foreach $file (@ARGV) {
    include_file($file, ('',));
}
```

Revision questions

The following questions are primarily intended for revision, either this week or later in session. Your tutor may still choose to cover some of these questions, time permitting.

1. Write a Perl program `source_count.pl` which prints the number of lines of C source code in the current directory. In other words, this Perl program should behave similarly to `wc -l *.[ch]`. (Note: you are not allowed to use `wc` or other Unix programs from within the Perl script). For example:

```
$ ./source_count.pl
   383 cyclorana.c
   280 cyclorana.h
    15 enum.c
   194 frequency.c
   624 model.c
   293 parse.c
   115 random.c
    51 smooth.c
   132 util.c
    16 util.h
   410 waveform.c
  2513 total
```

Sample Perl solution

```
#!/usr/bin/perl -w
# written by andrewt@cse.unsw.edu.au for COMP2041
# count lines of C source code
$total = 0;
foreach $file (glob("*.[ch]")) {
    open my $f, '<', $file or die "Can not open $file: $!";
    @lines = <$f>;
    $n_lines = @lines;
    # Why wouldn't   $n_lines = <FILE>;   work?

    printf "%7d %s\n", $n_lines, $file;
    $total += $n_lines;
    close $f;
}
printf "%7d total\n", $total;
```

Sample Python solution

```
#!/usr/bin/python
# written by andrewt@cse.unsw.edu.au for COMP2041
# count lines of C source code
import glob
total = 0
for filename in glob.glob("*.[ch]"):
    with open(filename) as f:
        lines = f.readlines()
        n_lines = len(lines)
        print("%7d %s" % (n_lines, filename))
        total += n_lines
print("%7d total"%total)
```

2. Write a Perl program, `cut.pl` which takes three arguments, Ö, N and a file name. It should print characters Ö-N of each line of the file. For example:

```
$ ./cut.pl 1 8 file
```

should print the 8 characters of every line in `file`.

Implement a version of the program which invokes `/usr/bin/cut` and a version which performs the operations directly in Perl.

Sample Perl solution calling /usr/bin/cut

```
#!/usr/bin/perl -w
system "cut -c$ARGV[0]-$ARGV[1] $ARGV[2]";
```

Sample solution in Perl itself

```
#!/usr/bin/perl -w
die "Usage: $0 <n> <m> <file" if @ARGV != 3;
open my $f, '<', $ARGV[2] or die "$0: can not open $ARGV[2]: $!";
while ($line = <$f>) {
    chomp $line;
    @chars = split //, $line;
    print @chars[$ARGV[0]-1..$ARGV[1]-1], "\n";
}
```

Sample solution implementing more of cut in Perl

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

$delim = "\t";
if ($ARGV[0] =~ /-d./) {
    ($delim = $ARGV[0]) =~ s/-d//;
    shift;
}
if ($ARGV[0] =~ /-f*/) {
    ($flist = $ARGV[0]) =~ s/-f//;
    if ($flist eq "")
        { shift; $flist = $ARGV[0]; }
    @fields = split(/,/,$flist);
    shift;
}
while (<>) {
    chomp;
    @words = split /$delim/;
    @outs = ();
    $nf = $#words;
    foreach $f (@fields) {
        push @outs, $words[$f-1] if ($f <= $nf+1);
    }
    print join($delim,@outs)."\n";
}
```

Python solution calling /usr/bin/cut

```python
#!/usr/bin/python
import subprocess, sys
print()
subprocess.call(["cut","-c%s-%s"%(sys.argv[1:3]), sys.argv[3]])
```

Sample solution in Python

```python
#!/usr/bin/python

import sys

if len(sys.argv) != 4:
    sys.stderr.write("Usage: %s <n> <m> <file>\n" % sys.argv[0])
    sys.exit(1)

for line in open(sys.argv[3]):
    line = line.rstrip('\n')
    print(line[int(sys.argv[1])-1:int(sys.argv[2])])
```

3. Implement a Perl script to solve the marks-to-grades problem that was solved as a shell script in a previous tutorial. Reminder: the script reads a sequence of (studentID,mark) pairs from its standard input and writes (studentID,grade) pairs to its standard output. The input pairs are written on a single line, separated by spaces, and the output should use a similar format. The script should also check whether the second value on each line looks like a valid grade, and output an appropriate message if it doesn't. The script can ignore any extra data occuring after the mark on each line.
Consider the following input and corresponding output to the program:

Input	Output
2212345 65 2198765 74 2199999 48 2234567 50 ok 2265432 99 2121212 hello 2222111 120 2524232 -1	2212345 CR 2198765 CR 2199999 FL 2234567 PS 2265432 HD 2121212 ?? (hello) 2222111 ?? (120) 2524232 ?? (-1)

Sample Perl solution

```
#!/usr/bin/perl -w
while (<>) {
    chomp;
    ($sid,$mark) = split;
    if ($mark !~ /^\d+$/) {
        $grade = "??";
    }
    else {
        $grade = "FL" if $mark >= 0 && $mark < 50;
        $grade = "PS" if $mark >= 50 && $mark < 65;
        $grade = "CR" if $mark >= 65 && $mark < 75;
        $grade = "DN" if $mark >= 75 && $mark < 85;
        $grade = "HD" if $mark >= 85 && $mark <= 100;
        $grade = "??" if $mark < 0 || $mark > 100;
    }
    $err = ($grade ne "??") ? "" : " ($mark)";
    print "$sid $grade$err\n";
}
```

4. Write a program `addressbook.pl` that reads two files `people.txt` and `phones.txt` containing data in CSV (comma-separated values) format and uses this data to print an address book in the format below:

```
$ cat people.txt
andrew,Andrew Taylor,42 Railway St,Petersham
arun,Arun Sharma,94 Leafy Close,Brisbane
gernot,Gernot Heiser,64 Trendy Tce,Newtown
jas,John Shepherd,16/256 Busy Rd,Alexandria
$ cat phones.txt
jas,home,9665 6432
arun,work,9385 5518
jas,work,9385 6494
arun,home,(07) 9314 6543
andrew,work,9385 4321
arun,mobile,0803 123 432
$ ./addressbook.pl
Andrew Taylor
42 Railway St, Petersham
Phones: 9385 4321 (work)

Arun Sharma
94 Leafy Close, Brisbane
Phones: 0803 123 432 (mobile), (07) 9314 6543 (home), 9385 5518 (work)
```

Assume that there are only three types of phone (mobile, home and work) and we always display them in that order.

ì ¢ÖÂ¢because the phone types are fixed, login name and phone type together can be the key used to look up a number. In this situation some suitable separator is used to create a composite, unambiguous key for the hash.

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

# Build hash table of phone numbers, where keys
# are "person:phone-type" strings

open my $phones, '<', "phones.txt" or die "Can not open phones.txt:$!\n";

while ($line = <$phones>) {
    chomp $line;
    my ($id,$type,$number) = split /,/, $line;
    $phones{"$id:$type"} = $number;
}

close $phones;

# Iterate through People, displaying values
open my $people, '<', "people.txt" or die "Can not open people.txt:$!\n";

while ($line = <$people>) {
    chomp $line;
    my ($id,$name,$street,$suburb) = split /,/, $line;
    print "$name\n$street, $suburb\nPhones: ";
    my $nphones = 0;
    foreach $type ('mobile', 'home', 'work') {
        my $key = "$id:$type";
        if (defined($phones{$key})) {
            $num = $phones{$key};
            print ", " if ($nphones++ > 0);
            print "$num ($type)";
        }
    }
    print "?" if ($nphones == 0);
    print "\n\n";
}

close $people;
```