# Welcome to Perl.

## Aims

Getting started with Perl programming.

## Assessment

**Submission:** `give cs2041 lab05 digits.pl echon.pl tail.pl`
also submit `shuffle.pl` and `shuffle_test.sh` if you attempt these challenge exercises

**Deadline:** either during the lab, or Sunday 30 August 11:59pm (midnight)

**Assessment:** Make sure that you are familiar with the lab assessment criteria (lab/assessment.html).

## Background

We have covered only a small amount of Perl in lectures. In fact, to cover the whole language in detail would take a whole semester, so we're going to rely on you finding out about the language yourself in tutes, labs and assignments. A good place to start is the Perl documentation & tutorial links on the class home page For example you might find these useful:

- Perl language syntax (http://search.cpan.org/dist/perl/pod/perlsyn.pod)
- Perl functions (http://search.cpan.org/dist/perl/pod/perlsub.pod)
- Perl operators (http://search.cpan.org/dist/perl/pod/perlop.pod)

## Storing lab work on gitlab.cse.unsw.edu.au

For this and future labs you are going to use a version control system named git to store copies of your lab work in a repository at gitlab.cse.unsw.edu.au.
Don't panic this is easy to do and will ensure you have a complete backup of all work on your lab 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

## Adding Your SSH Key to Gitlab

1. First print your CSE ssh key (if you have one:. This command should should do it.

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAyNSzIDylSPAAGLzU
```

2. If you couldn't print a ssh key with the above command, you need to generate an ssh key. You can do it like this (just hit return for each questions).

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/import/kamen/3
Created directory '/import/kamen/3/z5555555/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /import/kamen/
Your public key has been saved in /import/kamen/3/z5
The key fingerprint is:
b8:02:31:8b:bf:f5:56:fa:b0:1c:36:89:ad:e1:cb:ad z555
The key's randomart image is:
...
```
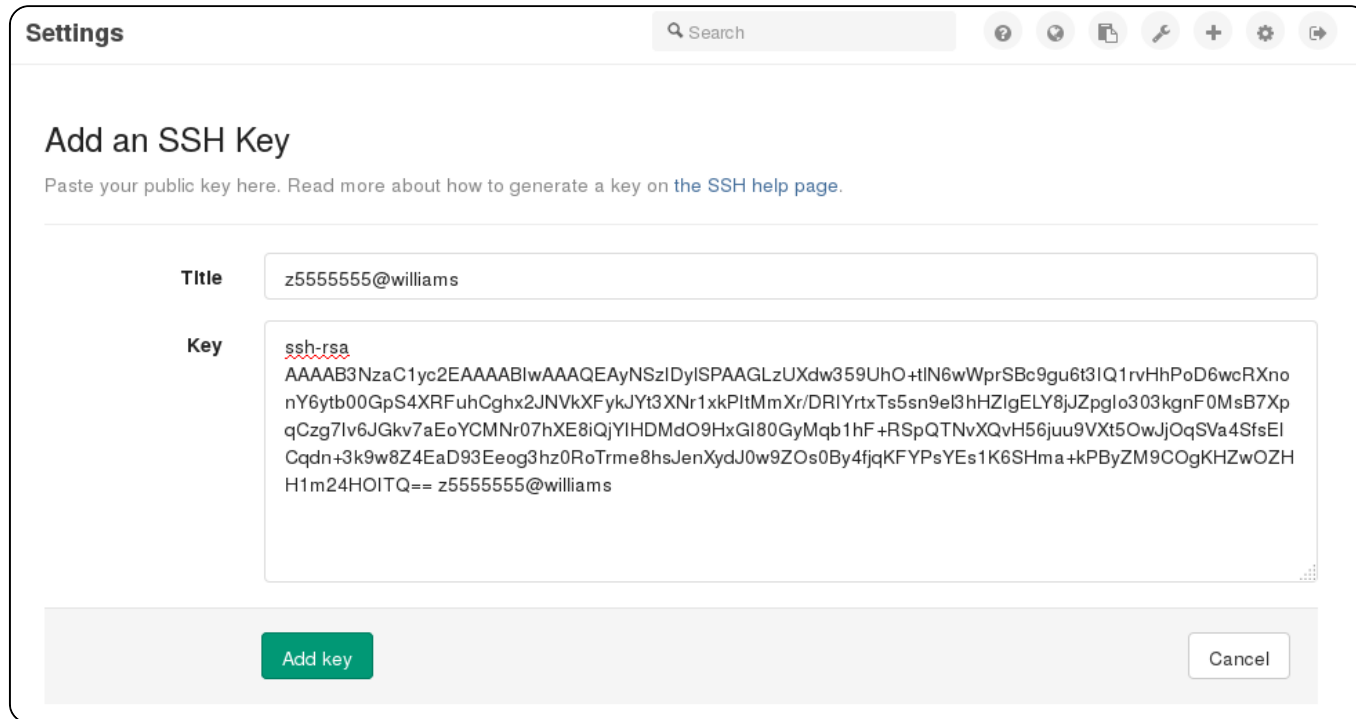
3. Now add your ssh key to gitlab:

4. Go to https://gitlab.cse.unsw.edu.au/profile/keys/new
   (https://gitlab.cse.unsw.edu.au/profile/keys/new) Click on [ Sign in ]

5. Supply your UNSW login (e.g. z5555555) and zPass and click the [ UNSW Sign in ] button.

6. Cut-and-paste your ssh-key (the entire 200+ character line printed by `cat ~/.ssh/id_rsa.pub` )
   into the "**Key**" field.
   Don't cut-and paste z5555555's ssh key above - cut-and-paste your ssh-key!

7. At this point, your screen should look something like this:

8. click the green <kbd>Add key</kbd> button

## Creating a Git Repository in your CSE account

A repository for your COMP[29]041 labs has already been created for you on `gitlab.cse.unsw.edu.au`.

You'll also need a git repository for your labs in your CSE account.

The instructions below will create a new directory for your COMP[29]041 labs named **2041-labs**.

From now on put the files for each lab in a sub-directory of this lab.

You can if you wish move earlier labs into this directory as well but do this **after** following the instructions below.

Make sure you replace *5555555* below by your student number!

```
$ cd
$ git clone gitlab@gitlab.cse.unsw.EDU.AU:z5555555/16s2-comp2041-labs 2041
Cloning into '2041-labs'...
$ chmod 700 2041-labs
$ cd 2041-labs
$ ln -sf /home/cs2041/public_html/scripts/autotest-pre-commit-hook .git/hooks/
$ ls -la
drwx------    3 z5555555  z5555555   4096 Aug 27 14:44 .
drwxr-x--x 107 z5555555  z5555555  16384 Aug 27 14:51 ..
drwx------    7 z5555555  z5555555   4096 Aug 27 14:44 .gi
drwx------    7 z5555555  z5555555   4096 Aug 27 14:44 lab
drwx------    7 z5555555  z5555555   4096 Aug 27 14:44 lab
...
$ cd lab05
```

You now have a git repository in your CSE account for this and future week's lab work. The sub-directory `.git` is where git stores information.

Create the files for this week's lab in **2041-labs/lab05** and push them to gitlab.cse.unsw.edu.au when every you make some progress.

BTW the line

```
$ ln -s /home/cs2041/public_html/16s2/scripts/autotest-pre-commit-hook .git/h
```

results in git running autotests for every commit. This should be useful to you and is small example of customizing git for building a software system.

# Exercise: Mapping Digits

Write a Perl script `digits.pl` that reads from standard input and writes to standard output mapping all digit characters whose values are less than 5 into the character '<' and all digit characters whose values are greater than 5 into the character '>'. The digit character '5' should be left unchanged.

| Sample Input Data | Corresponding Output |
|---|---|
| `1 234 5 678 9` | `< <<< 5 >>> >` |
| `I can think of 100's`<br>`of other things I'd rather`<br>`be doing than these 3 questions` | `I can think of <<<'s`<br>`of other things I'd rather`<br>`be doing than these < questions` |
| `A line with lots of numbers:`<br>`123456789123456789123456789`<br>`A line with all zeroes`<br>`000000000000000000000000000`<br>`A line with blanks at the end`<br>`1 2 3` | `A line with lots of numbers:`<br>`<<<<5>>>><<<<5>>>><<<<5>>>>`<br>`A line with all zeroes`<br>`<<<<<<<<<<<<<<<<<<<<<<<<<<<`<br>`A line with blanks at the end`<br>`< < <` |
| `Input with absolutely 0 digits in it`<br>`Well ... apart from that one ...` | `Input with absolutely < digits in it`<br>`Well ... apart from that one ...` |
| `1 2 4 8 16 32 64 128 256 512 1024`<br>`2048 4096 8192 16384 32768 65536` | `< < < > <> << >< <<> <5> 5<< <<<<`<br>`<<<> <<>> ><>< <><>< <<>>> >55<>` |

As usual you can run some tests on your script like this:

```
$ ~cs2041/bin/autotest lab05 digits.pl
```

Also do your own testing!

# Pushing to gitlab.cse.unsw.edu.au

When you make some progress with `digits.pl` do this to push it to gitlab.cse.unsw.edu.au.

```
$ git add digits.pl
$ git commit -a -m "first version"
[master 4cdfa5f] first version
 1 file changed, 17 insertions(+)
 create mode 100755 .pl
$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 239 bytes, done.
Total 2 (delta 1), reused 0 (delta 0)
To gitlab@calliope1.cse.unsw.EDU.AU:z5555555/16s2-comp2
    36ccb2b..4cdfa5f  master -> master
```

Later on when you have made more progress with `digits.pl` do this to commit the new version & push it to gitlab.

```
$ git commit -a -m "digits.pl passes autotests"
...
$ git push
```

If you explore at https://gitlab.cse.unsw.edu.au/z*5555555*/16s2-comp2041-labs (replacing *5555555* with your student number you'll find both versions of `digits.pl`

Follow the same steps for all the programs you write from now on for COMP[29]041.

# Exercise: Repeated Echo

Write a Perl script `echon.pl` which given exactly two arguments, an integer *n* and a string, prints the string *n* times. For example:

```
$ ./echon.pl 5 hello
hello
hello
hello
hello
hello
```

```
$ ./echon.pl 0 nothing
```

```
$ ./echon.pl 1 goodbye
goodbye
```

Your script should print an error message if it is not given exactly 2 arguments. For example:

```
$ ./echon.pl
Usage: ./echon.pl <number of lines> <string> at ./echon
```

```
$ ./echon.pl 1 2 3
Usage: ./echon.pl <number of lines> <string> at ./echon
```

As usual use `autotest` to assist in testing your code and push `echon.pl` to `gitlab.cse.unsw.edu.au` every time you make some progress with it.

```
$ ~cs2041/bin/autotest lab05 echo_n.pl
...
$ git add echon.pl
$ git commit -a -m "first version of echon.pl"
...
$ git push
...
```

## Exercise: Tail

Perl file manipulation

The standard approach in Perl for dealing with a collection of files whose names are supplied as command line arguments, is something like:

```perl
#!/usr/bin/perl -w
@files = ();
foreach $arg (@ARGV) {
    if ($arg eq "--version") {
        print "$0: version 0.1\n";
        exit(0);
    }
    # handle other options
    # ...
    else {
        push @files, $arg;
    }
}
foreach $f (@files) {
    open(F,"<$f") or die "$0: Can't open $f: $!\n";
    # process F
    #...
    close(F);
}
```

Write a Perl script to implement the Unix `tail` command. It should support the following features of `tail`:

- read from files supplied as command line arguments
- read from standard input if no file name arguments are supplied
- display the error message `tail.pl: can't open` *FileName* for any unreadable file
- display the last $N$ lines of each file (default $N = 10$)
- can adjust the number of lines displayed via an optional first argument $- N$
- if there are more than one named files, separate each by `==>` *FileName* `<==`

To assist with testing your solution, there are three small t files: t1.txt (lab/perl/tail/t1.txt), t2.txt (lab/perl/tail/t2.txt), and t3.txt (lab/perl/tail/t3.txt). Copy these files to your current directory.

```
$ cp /home/cs2041/public_html/lab/perl/tail/t?.txt .
```

Using these data files, your program should behave as follows:

```
$ ./tail.pl <t1.txt
Data 1 ... Line 2
Data 1 ... Line 3
Data 1 ... Line 4
Data 1 ... Line 5
Data 1 ... Line 6
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
 $ ./tail.pl t1.txt
Data 1 ... Line 2
Data 1 ... Line 3
Data 1 ... Line 4
Data 1 ... Line 5
Data 1 ... Line 6
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
 $ ./tail.pl -5 t1.txt
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
```

```
Data 1 ... Last line
$ ./tail.pl -5 t2.txt
A one line file.
$ ./tail.pl -5 t1.txt t2.txt t3.txt
==> t1.txt <==
Data 1 ... Line 7
Data 1 ... Line 8
Data 1 ... Line 9
Data 1 ... Line 10
Data 1 ... Last line
==> t2.txt <==
A one line file.
==> t3.txt <==
one
word
on
each
line
$ ./tail.pl -2 tX.txt
./tail.pl: can't open tX.txt
```

**Hint:** use the above template for Perl file processing to get started with your script. You *must* use the `-w` flag in your script, and you must write your code in such a way as to ensure that no warning messages are produced.

As usual use `autotest` to assist in testing your code and push `echon.pl` to `gitlab.cse.unsw.edu.au` every time you make some progress with it.

```
$ ~cs2041/bin/autotest lab05 tail.pl
...
$ git add tail.pl
$ git commit -a -m "initial tail.pl"
...
$ git push
...
```

## Challenge Exercise: Shuffling Lines

Write a Perl script `shuffle.pl` which prints its input with the lines in random order. For example:

```
$ i=0;while test $i -lt 5; do echo $i; i=$((i + 1)); done|./shuffle.pl
3
2
0
1
4
```

```
$ i=0;while test $i -lt 5; do echo $i; i=$((i + 1)); done|./shuffle.pl
1
0
3
4
2
```

You are not permitted to use `List::Util` (it contains a shuffle function).

Don't look for other people solutions - see if you can come up with your own. **Hint:** the perl function *rand* returns a floating point number between 0 and its argument. For example:

```
$ perl -e 'print rand(42), "\n"'
24.6945567464665
$ perl -e 'print rand(42), "\n"'
8.45308853250223
```

**Hint:** perl ignores the fractional part of a number if you use it to index an array

## Challenge Question: Testing Shuffling Lines

There is no dryrun test for `shuffle.pl`. Testing (pseudo)random programs is more difficult. because there are multiple correct outputs for a given input.

Write a shell script `shuffle_test.sh` which tests `shuffle.pl`.

Try to test that all outputs are correct and all correct outputs are being generated.

Don't forget to push `shuffle.pl` and `shuffle_test.sh` to `gitlab.cse.unsw.edu.au` if and when you work on them.

## Finalising

You must show your solutions to your tutor and be able to explain how they work. Once your tutor has discussed your answers with you, you should submit them using:

```
$ give cs2041 lab05 digits.pl echon.pl tail.pl [shuffle.pl shuffle_test.sh]
```

Whether you discuss your solutions with your tutor this week or next week, you must submit them before the above deadline.