Objectives

- Practicising Perl & Shell
- Exploring code useful for assignment 1

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 `lab08` by typing:

```
$ mkdir lab08
```

Change to this directory by typing:

```
$ cd lab08
```

Exercise: Backing Up a File

Write a Shell program, **backup.sh** which takes 1 argument, the name of a file.
Your program should create a backup copy of this file.

If the file is named **example.txt** the backup should be called **.example.txt.0** but you should not overwrite any previous backup copies.

So if **.example.txt.0** exists, the backup copy should be called **.example.txt.1** and if **.example.txt.1** also exists it should be called **.example.txt.2** and so on.

For example:

```
$ seq 1 3 >n.txt
$ cat n.txt
1
2
3
$ backup.sh n.txt
Backup of 'n.txt' saved as '.n.txt.0'
$ cat .n.txt.0
1
2
3
$ backup.sh n.txt
Backup of 'n.txt' saved as '.n.txt.1'
$ backup.sh n.txt
Backup of 'n.txt' saved as '.n.txt.2'
$ backup.sh n.txt
Backup of 'n.txt' saved as '.n.txt.3'
$ ls .n.txt.*
.n.txt.0
.n.txt.1
.n.txt.2
.n.txt.3
```

Your answer must be Shell. You can not use other languages such as Perl, Python or C.

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

```
$ 2041 autotest shell_backup
```

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

```
$ give cs2041 lab08_shell_backup backup.sh
```

Sample solution for `backup.sh`

```
#!/bin/bash

# backup a file given as a command line argument
# written by andrewt@unsw.edu.au as COMP[29]041 sample solution

for file in "$@"
do

    # go through .$file.0 .$file.1 .$file.2 , ...
    # looking for one that doesn't exist

    suffix=0
    while test -r ".$file.$suffix"
    do
        suffix=$((suffix + 1))
    done

    # make the backup

    if cp "$file" ".$file.$suffix"
    then
        echo "Backup of '$file' saved as '.$file.$suffix'"
    else
        echo "Backup of '$file' failed"
        exit 1
    fi
done
```

Exercise: A Perl Program to Back Up a File

Rewrite your shell script from the last exercise as a Perl program, **backup.pl** which takes 1 argument, the name of a file.
Your program should create a backup copy of this file.

If the file is named **example.txt** the backup should be called **.example.txt.0** but you should not overwrite any previous backup copies.

So if **.example.txt.0** exists, the backup copy should be called **.example.txt.1** and if **.example.txt.1** also exists it should be called **.example.txt.2** and so on.

For example:

```
$ seq 1 3 >n.txt
$ cat n.txt
1
2
3
$ backup.pl n.txt
Backup of 'n.txt' saved as '.n.txt.0'
$ cat .n.txt.0
1
2
3
$ backup.pl n.txt
Backup of 'n.txt' saved as '.n.txt.1'
$ backup.pl n.txt
Backup of 'n.txt' saved as '.n.txt.2'
$ backup.pl n.txt
Backup of 'n.txt' saved as '.n.txt.3'
$ ls .n.txt.*
.n.txt.0
.n.txt.1
.n.txt.2
.n.txt.3
```

Your answer must be Perl only. You can not use other languages such as Shell, Python or C.
You may not run external programs, e.g. via system or backquotes. for example, you can't run `cp`.

No error checking is necessary.

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

```
$ 2041 autotest perl_backup
```

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

```
$ give cs2041 lab08_perl_backup backup.pl
```

Sample solution for `backup.pl`

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

# backup a file given as a command line argument
# written by andrewt@unsw.edu.au as COMP[29]041 sample solution

sub main {
    for $file (@ARGV) {
        backup_file($file);
    }
}

# find an unused name for a backup copy
# and copy the file to that name

sub backup_file {
    my ($file) = @_;

    my $suffix = 0;

    # go through .$file.0 .$file.1 .$file.2 , ...
    # looking for one that doesn't exist

    while (-r ".$file.$suffix") {
        $suffix =$suffix + 1;
    }

    # make the backup

    copy_file($file,  ".$file.$suffix");
    print("Backup of '$file' saved as '.$file.$suffix'\n");
}

# could use File::Copy instead of writing our own function

sub copy_file {
    my ($source, $destination) = @_;

    open my $in, '<', $source or die "Cannot open $source: $!";
    open my $out, '>', $destination or die "Cannot open $destination: $!";

    while ($line = <$in>) {
        print $out $line;
    }

    close $in;
    close $out;
}

main()
```

Exercise: A Perl Program to Back Up a Directory

Write a Perl program, **snapshot.pl** which saves or restores backups of all the files in the current directory.
snapshot.pl will be called with a first argument of either **load** or **save**.

snapshot.pl save

If **snapshot.pl** is called with a first argument of **save** it should save copies of all files in the current directory.
snapshot.pl should first create a directory named **.snapshot.0** to store the backup copies of the files.

But if **.snapshot.0** already exists, the backup directory should be called **.snapshot.1** and if **.snapshot.1** also exists it should be called **.snapshot.2** and so on.

snapshot.pl should ignore files with names starting with .

snapshot.pl should also ignore itself (not backup snapshot.pl). **Hint**: Perl's `glob` and `mkdir` functions are useful.

snapshot.pl load *n*

If **snapshot.pl** is called with a first argument of **load** and a second argument of ***n*** it should restore (copy back) the files from snapshot **.snapshot.*n***.

Before doing this it should copy the current version of all files in a new **.snapshot** directory, in other words do the same as a **save** operation.

This is to make sure the user doesn't accidentally lose some work when restoring files. It is always done even if the user wouldn't lose work.

Examples

```
$ ls .snapshot.*/*
ls: cannot access .snapshot.*/*: No such file or directory
$ echo hello >a.txt
$ snapshot.pl save
Creating snapshot 0
$ ls .snapshot.*/*
.snapshot.0/a.txt
$ echo word >a.txt
$ snapshot.pl load 0
Creating snapshot 1
Restoring snapshot 0
$ ls .snapshot.*/*
.snapshot.0/a.txt
.snapshot.1/a.txt
$ cat a.txt
hello
```

and

```
$ echo hello0 >a.txt
$ echo world0 >b.txt
$ snapshot.pl save
Creating snapshot 0
$ echo hello1 >a.txt
$ echo world1 >b.txt
$ snapshot.pl save
Creating snapshot 1
$ echo hello2 >a.txt
$ echo world2 >b.txt
$ ls .snapshot.*/*
.snapshot.0/a.txt
.snapshot.0/b.txt
.snapshot.1/a.txt
.snapshot.1/b.txt
$ snapshot.pl load 0
Creating snapshot 2
Restoring snapshot 0
$ grep . ?.txt
a.txt:hello0
b.txt:world0
$ snapshot.pl load 1
Creating snapshot 3
Restoring snapshot 1
$ grep . ?.txt
```

Your answer must be Perl only. You can not use other languages such as Shell, Python or C.
You may not run external programs, e.g. via system or backquotes. for example, you can't run `cp`.

No error checking is necessary.

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

```
$ 2041 autotest snapshot
```

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

```
$ give cs2041 lab08_snapshot snapshot.pl
```

Sample solution for `snapshot.pl`

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

# backup a file given as a command line argument
# written by andrewt@unsw.edu.au as COMP[29]041 sample solution

sub main() {
    if (@ARGV == 1 && $ARGV[0] eq "save") {
        save_snapshot();
    } elsif (@ARGV == 2 && $ARGV[0] eq "load") {
        save_snapshot();
        load_snapshot($ARGV[1]);
    } else {
        usage();
    }
}

sub usage {
    print <<eof;;
Usage snapshot.pl <command>
Commands:
save       Creates a snapshot of the current directory
load n     Loads the n'th snapshot into the current directory
eof
}

# copy all files in the current directory (unless they start with .)
# to the snapshort directory

sub save_snapshot {
    my $snapshot_directory = create_new_snapshot_directory();

    for $file (glob  "*") {
        next if $file eq "snapshot.pl";
        copy_file($file, "$snapshot_directory/$file");
    }
}

# copy all files from the snapshort directory to the current directory

sub load_snapshot {
    my ($suffix) = @_;
    my $snapshot_directory = ".snapshot.$suffix";

    die "Unknown snapshot $suffix" if ! -d $snapshot_directory;

    print "Restoring snapshot $suffix\n";

    for $snapshot_file (glob "$snapshot_directory/*") {
        my $file = $snapshot_file;
        $file =~ s/.*\///;
        copy_file($snapshot_file, $file);
    }
}

# find an unused name for a snapshot directory
# create a directory of that name, and return it

sub create_new_snapshot_directory {
    my $suffix = 0;
    while (1) {
        my $snapshot_directory = ".snapshot.$suffix";

        if (!-d $snapshot_directory) {
            mkdir $snapshot_directory or die "can not create $snapshot_directory: $!\n";
            print "Creating snapshot $suffix\n";
            return $snapshot_directory;
        }

        $suffix = $suffix + 1;
    }
}

# could use File::Copy instead of writing our own function
```

```
sub copy_file {
    my ($source, $destination) = @_;

    open my $in, '<', $source or die "Cannot open $source: $!";
    open my $out, '>', $destination or die "Cannot open $destination: $!";

    while ($line = <$in>) {
        print $out $line;
    }

    close $in;
    close $out;
}

main()
```

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 **Sunday 16 September 23:59:59** 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.