Objectives

- Practising manipulating files with Perl & Shell
- Understanding Perl & Shell strenths & weaknesses
- 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 `lab05` and change to this directory with these comamnds:

```
$ mkdir lab05
$ cd lab05
```

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
```

Autotest Results

99% of **501** students who have autotested **backup.sh** so far, passed all autotest tests.

- **100%** passed test *autotest0*
- **99%** passed test *autotest1*
- **99%** passed test *autotest2*

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

```
$ give cs2041 lab05_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
```

Autotest Results

99% of **487** students who have autotested **backup.pl** so far, passed all autotest tests.

- **99%** passed test *autotest0*
- **99%** passed test *autotest1*
- **99%** passed test *autotest2*

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

```
$ give cs2041 lab05_perl_backup backup.pl
```

Sample solution for `backup.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 {
    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: Shell Programs to Back Up a Directory

Write Shell scripts **snapshot-save.sh** & **snapshot-load.sh** which saves & restore backups of all the files in the current directory. These scripts should be in Posix-compatible Shell, use:

```
#!/bin/dash
```

snapshot-save.sh

If **snapshot-save.sh** should save copies of all files in the current directory.
snapshot-save.sh 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-save.sh should ignore files with names starting with .

snapshot-save.sh should also ignore itself and snapshot-load.sh (not backup snapshot-save.sh and snapshot-load.sh).

snapshot-load.sh *n*

If **snapshot-load.sh** is called with a first 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, (hint run **snapshot-save.sh**)

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

Your answer must be Posix-compatible Shell only. You can not use Bash ,Perl, Python or C.
No error checking is necessary.

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

```
$ 2041 autotest shell_snapshot
```

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

```
$ give cs2041 lab05_shell_snapshot snapshot-save.sh snapshot-load.sh
```

Sample solution for `snapshot-save.sh`

```
#!/bin/dash

if test $# != 0
then
    echo "Usage: $0" 1>&2
    exit 1
fi


suffix=0
while test -e ".snapshot.$suffix"
do
    suffix=$((suffix + 1))
done

echo "Creating snapshot $suffix"

snapshot_directory=".snapshot.$suffix"
mkdir $snapshot_directory || exit 1

for file in *
do
    if test -f "$file" -a "$file" != snapshot-save.sh -a "$file" != snapshot-load.sh
    then
        cp -p "$file" "$snapshot_directory"
    fi
done
```

Sample solution for `snapshot-load.sh`

```
#!/bin/dash

if test $# != 1
then
    echo "Usage: $0 <snapshot-number>" 1>&2
    exit 1
fi

suffix=$1

snapshot_directory=".snapshot.$suffix"

if test ! -d $snapshot_directory
then
    echo "Unknown snapshot $suffix" 1>&2
    exit 1
fi

snapshot-save.sh

echo "Restoring snapshot $suffix"

cp -p $snapshot_directory/* .
```

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 perl_snapshot
```

Autotest Results

97% of **442** students who have autotested **snapshot.pl** so far, passed all autotest tests.

- **97%** passed test *1*
- **97%** passed test *2*

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

```
$ give cs2041 lab05_perl_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()
```

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
```

Autotest Results

63% of **155** students who have autotested **perl_print.pl** so far, passed all autotest tests.

- **81%** passed test *0*
- **81%** passed test *1*
- **81%** passed test *2*
- **73%** passed test *3*
- **72%** passed test *4*
- **72%** passed test *5*
- **72%** passed test *6*

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

```
$ give cs2041 lab05_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";
```

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 **Wednesday 10 July 21:59:59** 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 1.2 marks.

Usually each lab exercise will be worth the same - for example if there are 5 lab exercises each will be worth 0.4 marks.

Except challenge exercises (see below) will never total more than 20% of each week's lab mark.

All of your lab marks for weeks 1-10, will be summed to give you a mark out of 12.

If their sum exceeds 9 - your total mark will be capped at 9.

Running Autotests On your Own Computer

An experimental version of autotest exists which may allow you to run autotest on your own computer.
If you are running Linux, Windows Subsystem for Linux or OSX. These commands might let you run autotests at home.

```
$ sudo wget https://cgi.cse.unsw.edu.au/~cs2041/19T2/resources/home_autotest -O/usr/local/bin/2041_autotest
$ sudo chmod 755 /usr/local/bin/2041_autotest
$ 2041_autotest shell_snapshot
```

Autotest itself needs Python 3.6 (or later) installed.
Particular autotests may require other software install, e.g. autotests of perl programs require Perl installed (of course).

The legit autotests need python3.7, git & binfmt-support installed.

The program embeds the autotests themselves, so you'll need to re-download if autotests are changed, added, fixed, ...

If it breaks on your computer post on the class forum and we'll fix if we can, but this is very definitely experimental.