

Week 05 Laboratory Sample Solutions

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`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab05
$ cd lab05
$ 2041 fetch lab05
```

Or, if you're not working on CSE, you can download the provided code as a [zip file](#) or a [tar file](#).

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

```

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 lab05_shell_backup backup.sh
```

before **Tuesday 14 July 21:00** to obtain the marks for this lab exercise.

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
```

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 lab05_perl_backup backup.pl
```

before **Tuesday 14 July 21:00** to obtain the marks for this lab exercise.

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
```

Your answer must be Posix-compatible Shell only. You can not use Bash ,Perl, Python or C.

Autotest and automarking will run your scripts with a current working directory different to the directory containing the script. The directory containing your submission will be in \$PATH.

This means ./snapshot-save.sh run from snapshot-load.sh, but running snapshot-save.sh will succeed.

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
```

before **Tuesday 14 July 21:00** to obtain the marks for this lab exercise.

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
```

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
```

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

```
$ give cs2041 lab05_perl_snapshot snapshot.pl
```

before **Tuesday 14 July 21:00** to obtain the marks for this lab exercise.

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 snapshot 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 snapshot 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
```

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

```
$ give cs2041 lab05_perl_print perl_print.pl
```

before **Tuesday 14 July 21:00** to obtain the marks for this lab exercise.

Sample solution for perl_print.pl

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

# Output a Perl program which when run will print the
# string supplied as a cmdline 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 **Tuesday 14 July 21:00** to submit your work.

You cannot obtain marks by e-mailing your code 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, using test cases different to those autotest runs for you. (Hint: do your own testing as well as running autotest.)

After automarking is run by the lecturer you can [view your results here](#). The resulting mark will also be available [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
```

COMP(2041|9044) 20T2: 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