COMP(2041|9044) 19T2 **Assignment 1 - Legit** COMP(2041|9044) 19T2

version: 0.8 last updated: 2019-07-08 09:30

Aims

This assignment aims to give you

- practice in Shell programming generally
- introduce you to git & give you a clear concrete understanding of its basic semantics

Note: the material in the lecture notes will not be sufficient by itself to allow you to complete this assignment. You may need to search on-line documentation for Shell, git, Perl etc. Being able to search documentation efficiently for the information you need is a *very* useful skill for any kind of computing work.

Introduction

Your task in this assignment is to write Shell & Perl programs which implement a subset of the version control system Git.
Git is a very complex program which has many individual commands. You will implement only a few of the most important commands.

You will be given a number of simplifying assumptions which make your task easier.

Reference implementation

Many aspects of this assignment are not fully specified in this document.
Instead you must match the behaviour of reference implementations.

For example your script **legit-add** should match the behaviour of **2041 legit-add** exactly, including producing the same error messages.

Provision of a reference implementation is a common method to provide an operational specification, and it's something you will likely need to do after you leave UNSW.

Discovering & matching the reference implementation's behaviour is deliberately part of the assignment.

While the code in the reference implementation is fairly straight forward, reverse-engineering its behaviour is obviously not so simple and it's a nice example of how coming to grips with the precise semantics of an apparently obvious task can still be challenging.

If you discover what you believe to be a bug in the reference implementation, report it in the class forum. Andrew may fix the bug or indicate that you do not need to match the reference implementation's behaviour in this case.

Legit Commands - Subset 0

Subset 0 commands must be implemented in POSIX-compatible Shell. See the **Permitted Languages** section for more information.

legit-init

The **legit-init** command creates an empty Legit repository.
legit-init should create a directory named **.legit** which it will use to store the repository.
It should produce an error message if this directory already exists. You should match this and other error messages exactly. For example:

```
$ ls -d .legit
ls: cannot access '.legit': No such file or directory
$ legit-init
Initialized empty legit repository in .legit
$ ls -d .legit
.legit
$ legit-init
legit-init: error: .legit already exists
```

legit-init may create initial files or directories inside **.legit**.
You do not have to use a particular representation to store the repository.

You do not have to create the same files or directory inside **legit-init** as the reference implementation.

legit-add *filenames*

The **legit-add** command adds the contents of one or more files to the "**index**".
Files are added to the repository in a two step process. The first step is adding them to the index.

You will need to store files in the index somehow in the **.legit** sub-directory. For example you might choose store them in a sub-directory of **.legit**.

Only ordinary files in the current directory can be added, and their names will always start with an alphanumeric character ([a-zA-Z0-9]) and will only contain alpha-numeric characters plus '.', '-' and '_' characters.

The **legit-add** command, and other **legit** commands, will not be given pathnames with slashes.

legit-commit -m *message*

The **legit-commit** command saves a copy of all files in the index to the repository.
A message describing the commit must be included as part of the commit command.

legit commits are numbered (not hashes like git). You must match the numbering scheme.

You can assume the commit message is ASCII, does not contain new-line characters and does not start with a '-' character.

legit-log

The **legit-log** command prints one line for every commit that has been made to the repository.
Each line should contain the commit number and the commit message.

legit-show commit:filename

The **legit-show** should print the contents of the specified file as of the specified commit.
If the commit is omitted the contents of the file in the index should be printed.

For example:

```
$ ./legit-init
Initialized empty legit repository in .legit
$ echo line 1 > a
$ echo hello world >b
$ ./legit-add a b
$ ./legit-commit -m 'first commit'
Committed as commit 0
$ echo  line 2 >>a
$ ./legit-add a
$ ./legit-commit -m 'second commit'
Committed as commit 1
$ ./legit-log
1 second commit
0 first commit
$ echo line 3 >>a
$ ./legit-add a
$ echo line 4 >>a
$ ./legit-show 0:a
line 1
$ ./legit-show 1:a
line 1
line 2
$ ./legit-show :a
line 1
line 2
```

Legit Commands - Subset 1

Subset 1 is more difficult and you will need spend some time understanding the semantics (meaning) of these operations by running the reference implementation or researching the equivalent git operations.
Note the assessment scheme recognizes this difficulty.

Subset 1 commands must be implemented in POSIX-compatible Shell. See the **Permitted Languages** section for more information.

legit-commit [-a] -m *message*

legit-commit can have a **-a** option which causes all files already in the index to have their contents from the current directory added to the index before the commit.

legit-rm [--force] [--cached] *filenames*

legit-rm removes a file from the index, or from the current directory and the index.
If the **--cached** option is specified the file is removed only from the index and not from the current directory.

legit-rm like **git rm** should stop the user accidentally losing work, and should give an error message instead of if the removal would cause the user to lose work.

You will need to experiment with the reference implementation to discover these error messages. Researching **git rm**'s behaviour may also help.

The **--force** option overrides this, and will carry out the removal even if the user will lose work.

legit-status

legit-status shows the status of files in the current directory, index, and repository.

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

Legit Commands - Subset 2

Subset 2 is extremely difficult and you will need spend considerable time understanding the semantics of these operations by running the reference implementation and researching the equivalent git operations.
Note the assessment scheme recognizes this difficulty.

Subset 2 commands must be implemented in Perl. See the **Permitted Languages** section for more information.

legit-branch [-d] [*branch-name*]

legit-branch either creates a branch, deletes a branch or lists current branch names.

legit-checkout *branch-name*

legit-checkout switches branches.
Note unlike **git** you can not specify a commit or a file, you can only specify a branch.

legit-merge *branch-name*|*commit* -m message

legit-merge -m *message* adds the changes that have been made to the specified branch or commit to the index and commits them.

```
$ ./legit-init
Initialized empty legit repository in .legit
$ seq 1 7 >7.txt
$ ./legit-add 7.txt
$ ./legit-commit -m commit-1
Committed as commit 0
$ ./legit-branch b1
$ ./legit-checkout b1
Switched to branch 'b1'
$ perl -pi -e 's/2/42/' 7.txt
$ cat 7.txt
1
42
3
4
5
6
7
$ ./legit-commit -a -m commit-2
Committed as commit 1
$ ./legit-checkout master
Switched to branch 'master'
$ cat 7.txt
1
2
```

If a file contains conflicting changes **legit-merge** produces an error message.

```
$ ./legit-init
Initialized empty legit repository in .legit
$ seq 1 7 >7.txt
$ ./legit-add 7.txt
$ ./legit-commit -m commit-1
Committed as commit 0
$ ./legit-branch b1
$ ./legit-checkout b1
Switched to branch 'b1'
$ perl -pi -e 's/2/42/' 7.txt
$ cat 7.txt
1
42
3
4
5
6
7
$ ./legit-commit -a -m commit-2
Committed as commit 1
$ ./legit-checkout master
Switched to branch 'master'
$ cat 7.txt
1
2
```

Diary

You must keep notes on each piece of work you make on this assignment. The notes should include date, starting & finishing time, and a brief description of the work carried out. For example:

Date	Start	Stop	Activity	Comments
19/06/19	16:00	17:30	coding	implemented basic commit functionality
20/06/19	20:00	10:30	debugging	found bug in command-line arguments

Include these notes in the files you submit as an ASCII file named diary.txt.

Testing

As usual some autotests will be available:

```
$ 2041 autotest legit legit-*
...
```

You can also run only tests for a particular subset or an individual test:

```
$ 2041 autotest legit subset1 legit-*
...
$ 2041 autotest legit subset1_13 legit-*
...
```

If you are using extra Perl or Shell files, include them on the autotest command line.
You will need to do most of the testing yourself.

Test Scripts

You should submit ten Shell scripts named `test00.sh .. test09.sh` which run legit commands that test an aspect of Legit.
The **test??.sh** scripts do not have to be examples that your program implements successfully.

You may share your test examples with your friends but the ones you submit must be your own creation.

The test scripts should show how you've thought about testing carefully.

Permitted Languages

The subset 0 & 1 commands (legit-init, legit-add, legit-commit, legit-show, legit-rm, legit-status) must be written entirely in POSIX-compatible shell.
Your programs will be run with /bin/dash.

Start your programs with:

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

If you want run your scripts on your own machine (e.g. running OSX) which has dash installed somewhere other than /bin use:

```
#!/usr/bin/env dash
```

You are permitted to use any feature `/bin/dash` provides.
On CSE system /bin/sh is the Bash Shell (/bin/sh is a symlink to /bin/bash). The Bash Shell implements many non-POSIX extensions including regular expressions and arrays. These will not work with /bin/dash. You are not permitted to use these for the assignment.

You are not permitted to use Perl, Python or any other language except POSIX-compatible shell for subsets 0 & 1.

You are permitted to use only these external programs:

basename bunzip2 bzcat bzip2 cat chmod cmp cp cpio csplit cut date dc dd df diff dirname du echo egrep env expand expr false fgrep find fold getopt grep gunzip gzip head hostname less ln ls lzcat lzma md5sum mkdir mktemp more mv patch printf pwd readlink realpath rev rm rmdir sed seq sha1sum sha256sum sha512sum sleep sort stat strings tac tail tar tee test time top touch tr true uname uncompress unexpand uniq unlzma unxz unzip wc wget which who xargs xz xzcat yes zcat

Only a few of the programs in the above list are likely to be useful for the assignment.

If you wish to use an external program for subset 0 or 1 which is not in the above list, please ask in the class forum for it to be added.

You may submit extra Shell files.

The subset 2 commands (legit-branch, legit-checkout, legit-merge) must be entirely written in Perl.

They may not run external programs (e.g. via `system` or back-quotes). One exception is that legit-merge may run legit-commit.

They should run with version of Perl installed on CSE lab machines.

They will be run with Perl's **-w** flag and should not produce any warnings.

You may only use Perl packages which are installed on CSE's lab computers.

You may submit extra Perl files.

Assumptions/Clarifications

Like all good programmers, you should make as few assumptions as possible.
You can assume **legit** commands are always run in the same directory as the repository and only files from that directory are added to the repository.

You can assume that the directory in which **legit** commands are run will not contain sub-directories apart from .legit.

You can assume that branch names all start with an alphanumeric character ([a-zA-Z0-9]) and will only contain alphanumeric characters plus '-' and '_'. Also, branch names cannot be entirely numeric, so that they can be distinguished from commits when merging.

You can assume that (legit-add, legit-show, legit-rm) will be given just a filename, not pathnames with slashes.

You do not have to consider file permissions or other file metadata, for example you do not have to ensure files created by a checkout command have the same permisisons as when they were added.

You do not have to handle concurrency. You can assume only one instance of any **legit** command is running at any time.

You assume only the arguments described above are supplied to legit commands. You do not have to handle other arguments.

You should match the output streams used by the reference implementations. It writes error messages to stderr, so should you.

You should match the exit status used by the reference implementation. It exits with status 1 after an error, so should you.

Autotest and automarking will run your scripts with a current working directory different to the directory containing the script. This may break Shell or Perl with code in extra files, if so ask for help in the forum. The directory containing your submission will be in $PATH.

You can assume arguments will be the position and order shown in the usage message from the reference implementation.

Other orders and positions will not be tested.

For example here is the usage message for **legit-rm**:

```
$ 2041 legit-rm
usage: legit-rm [--force] [--cached] <filenames>
```

So you assume that if the **--force** or **--cached** options are present they come before all filenames and if they are both present the **--force** option will come first.

Change Log

Version 0.1
(2019-06-19 19:30)
- Initial release

Version 0.2
(2019-06-27 11:30)
- additional information about #! line added

Version 0.2
(2019-06-28 09:00)
- additional assumption about arguments added

Version 0.3
(2019-06-29 12:00)
- csplit added to list of permitted programs

Version 0.3
(2019-06-29 12:00)
- exception added for running legit-commit from legit-merge

Version 0.4
(2019-06-30 21:30)
- corrected legit-init example

Version 0.5
(2019-06-30 10:14)
- corrected legit-rm example

Version 0.6
(2019-06-30 10:14)
- corrected legit-rm description in spec

Version 0.7
(2019-07-08 09:30)
- corrected order of argument for legit-merge in autotest to match spec
- correct type in referenece implementation error message s/depository/repositiory/

Version 0.8
(2019-07-08 09:30)
- corrected indicative marking scheme to remove reference to non-existent challenge component

Attribution of Work

This is an individual assignment. The work you submit must be your own work and only your work apart from any exceptions explicitly included in the assignment specification above.
Joint work is not permitted.

You are only permitted to request help with the assignment in the course forum, help sessions or from course lecturers or tutors.

Do not provide or show your assignment work to any other person (including by posting it on the forum) apart from the teaching staff of COMP(2041|9044).

The work you submit must otherwise be entirely your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

We are required to inform scholarship authorities if students holding scholarships are involved in an incident of plagiarism or other misconduct, and this may result in a loss of the scholarship.

Plagiarism or other misconduct can also result in loss of student visas.

If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Note, you will not be penalized if your work is taken without your consent or knowledge.

Submission of Work

You are required to submit intermediate versions of your assignment.
Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the give command below.

It is fine if intermediate versions do not compile or otherwise fail submission tests.

Only the final submitted version of your assignment will be marked.

All these intermediate versions of your work will be placed in a git repository and made available to you via a web interface at this URL, replace *z5555555* with your own zid: `https://gitlab.cse.unsw.edu.au/z5555555/19T2-comp2041-ass1_legit/commits/master`

This will allow you to retrieve earlier versions of your code if needed.

You submit your work like this:

```
$ give cs2041 ass1_legit legit-* diary.txt test??.sh [any-other-files]
```

Assessment

This assignment will contribute 15 marks to your final COMP(2041|9044) mark
15% of the marks for assignment 1 will come from hand marking. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, you will be assessed on how easy it is for a human to read and understand your program.

5% of the marks for assignment 1 will be based on the test suite you submit.

80% of the marks for assignment 1 will come from the performance of your code on a large series of tests.

HD+ 100	All subsets working, code is beautiful, great test suite & diary
DN (80)	Subset 1 working, good clear code, good test suite & diary
CR (70)	Subset 0 working, good clear code, good test suite & diary
PS (60)	Subset 0 passing some tests, code is reasonably readable, reasonable test suite & diary
PS (55)	Subset 0 working internally (storing data) put not passing tests
0%	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP(2041|9044)	Submitting any other person's work. This includes joint work.

academic misconduct	Submitting another person's work without their consent. Paying another person to do work for you.

The lecturer may vary the assessment scheme after inspecting the assignment submissions but its likely to be broadly similar to the above.

Due Date

This assignment is tentatively due Sunday July 14 21:59
If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 2%. For example if an assignment worth 74% was submitted 10 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late it would be awarded 70%, the maximum mark it can achieve at that time.

COMP(2041|9044) 19T2: 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