Assignment 1: Pushy

version: 1.0 last updated: 2024-03-06 12:00

Aims

This assignment aims to give you

practice in Shell programming generally

a clear concrete understanding of Git's core 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, etc. Being able to search documentation efficiently for the information you need is a very useful skill for any kind of computing work.

Introduction

You are going to implement Pushy, a simple but powerful subset of the version control system Git.

Your task in this assignment is to write 10 shell scripts named pushy-init pushy-add pushy-commit pushy-log pushy-show pushy-rm pushy-status pushy-branch pushy-checkout pushy-merge.

Each of these script implements a simplified version of the corresponding Git command.

Git is a very complex program that has many individual commands. You only have to implement simplified equivalents of some core commands.

You are given a number of simplifying assumptions which make your task much easier.

Interestingly, large parts of early versions of Git were implemented in Shell.

Reference implementation

Many aspects of this assignment are not fully specified in this document; instead, you must match the behaviour of a reference implementation.

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

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

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

While the code in the reference implementation is fairly straightforward, reverse-engineering its behaviour is obviously not so simple, and is 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. We may fix the bug, or indicate that you do not need to match the reference implementation's behaviour in this case.

Pushy Commands

Subset 0

Subset 0 commands must be implemented in POSIX-compatible Shell.

See the Permitted Languages section for more information.

pushy-init

The pushy-init command creates an empty Pushy repository.

pushy-init should create a directory named .pushy, which it will use to store the repository.

It should produce an error message if this directory already exists, or cannot be created.

You should match this, and other error messages exactly. For example:

ls -d .pushy

ls: cannot access '.pushy': No such file or directory

./pushy-init

/bin/sh: 1: ./pushy-init: not found

ls -d .pushy

ls: cannot access '.pushy': No such file or directory

./pushy-init

/bin/sh: 1: ./pushy-init: not found

pushy-init may create initial files or directories inside .pushy.

You do not have to use a particular representation to store the repository.

You do not have to (and should not) create the same files and directories inside .pushy as the reference implementation.

You can create whatever files or directories inside .pushy you wish.

Do not store information outside .pushy

pushy-add filenames...

The pushy-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 .pushy sub-directory.

For example, you might choose store them in a sub-directory of .pushy.

Only ordinary files in the current directory can be added.

You can assume filenames start with an alphanumeric character ([a-zA-Z0-9]) and will only contain alpha-numeric characters, plus ., - and \_ characters.

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

pushy-commit -m message

The pushy-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.

Pushy commits are numbered sequentially: they are 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.

pushy-log

The pushy-log command prints a line for every commit made to the repository.

Each line should contain the commit number and the commit message.

pushy-show [commit]:filename

The pushy-show should print the contents of the specified filename as of the specified commit.

If commit is omitted, the contents of the file in the index should be printed.

You can assume the commit, if specified, will be a non-negative integer.

Subset 0 examples

./pushy-init

/bin/sh: 1: ./pushy-init: not found

echo line 1 > a

echo hello world >b

./pushy-add a b

/bin/sh: 1: ./pushy-add: not found

./pushy-commit -m 'first commit'

/bin/sh: 1: ./pushy-commit: not found

echo line 2 >>a

./pushy-add a

/bin/sh: 1: ./pushy-add: not found

./pushy-commit -m 'second commit'

/bin/sh: 1: ./pushy-commit: not found

./pushy-log

/bin/sh: 1: ./pushy-log: not found

echo line 3 >>a

./pushy-add a

/bin/sh: 1: ./pushy-add: not found

echo line 4 >>a

./pushy-show 0:a

/bin/sh: 1: ./pushy-show: not found

./pushy-show 1:a

/bin/sh: 1: ./pushy-show: not found

./pushy-show :a

/bin/sh: 1: ./pushy-show: not found

cat a

line 1

line 2

line 3

line 4

./pushy-show 0:b

/bin/sh: 1: ./pushy-show: not found

./pushy-show 1:b

/bin/sh: 1: ./pushy-show: not found

Subset 1

Subset 1 is more difficult. You will need to 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 recognises this difficulty.

Subset 1 commands must be implemented in POSIX-compatible Shell.

See the Permitted Languages section for more information.

pushy-commit [-a] -m message

pushy-commit can now 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.

pushy-rm [--force] [--cached] filenames...

pushy-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.

pushy-rm, like git rm, should stop the user accidentally losing work, and should give an error message instead 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.

pushy-status

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

There are many different cases to consider for pushy-status.

You will need to experiment with the reference implementation to find them all.

Subset 1 examples

./pushy-init

/bin/sh: 1: ./pushy-init: not found

touch a b c d e f g h

./pushy-add a b c d e f

/bin/sh: 1: ./pushy-add: not found

./pushy-commit -m 'first commit'

/bin/sh: 1: ./pushy-commit: not found

echo hello >a

echo hello >b

./pushy-commit -a -m 'second commit'

/bin/sh: 1: ./pushy-commit: not found

echo world >>a

echo world >>b

echo hello world >c

./pushy-add a

/bin/sh: 1: ./pushy-add: not found

echo world >>b

rm d

./pushy-rm e

/bin/sh: 1: ./pushy-rm: not found

./pushy-add g

/bin/sh: 1: ./pushy-add: not found

./pushy-status

/bin/sh: 1: ./pushy-status: not found

Subset 2

Subset 2 is extremely difficult. You will need to spend considerable time understanding the semantics of these operations, by running the reference implementation, and/or researching the equivalent Git operations.

Note the assessment scheme recognises this difficulty.

Subset 2 commands must be implemented in POSIX-compatible Shell.

See the Permitted Languages section for more information.

pushy-branch [-d] [branch-name]

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

If branch-name is omitted, the names of all branches are listed.

If branch-name is specified, then a branch with that name is created or deleted,

depending on whether the -d option is specified.

pushy-checkout branch-name

pushy-checkout switches branches.

Note that, unlike Git, you can not specify a commit or a file: you can only specify a branch.

pushy-merge (branch-name|commit-number) -m message

pushy-merge adds the changes that have been made to the specified branch or commit to the index, and commits them.

Subset 2 examples

./pushy-init

/bin/sh: 1: ./pushy-init: not found

seq 1 7 >7.txt

./pushy-add 7.txt

/bin/sh: 1: ./pushy-add: not found

./pushy-commit -m commit-1

/bin/sh: 1: ./pushy-commit: not found

./pushy-branch b1

/bin/sh: 1: ./pushy-branch: not found

./pushy-checkout b1

/bin/sh: 1: ./pushy-checkout: not found

sed -Ei 's/2/42/' 7.txt

cat 7.txt

1

42

3

4

5

6

7

./pushy-commit -a -m commit-2

/bin/sh: 1: ./pushy-commit: not found

./pushy-checkout master

/bin/sh: 1: ./pushy-checkout: not found

cat 7.txt

1

42

3

4

5

6

7

./pushy-merge b1 -m merge-message

/bin/sh: 1: ./pushy-merge: not found

cat 7.txt

1

42

3

4

5

6

7

If a file has been changed in both branches pushy-merge produces an error message.

Note: if a file has been changed in both branches git examines which lines have been changed and combines the changes if possible. Pushy doe not do this, for example:

./pushy-init

/bin/sh: 1: ./pushy-init: not found

seq 1 7 >7.txt

./pushy-add 7.txt

/bin/sh: 1: ./pushy-add: not found

./pushy-commit -m commit-1

/bin/sh: 1: ./pushy-commit: not found

./pushy-branch b1

/bin/sh: 1: ./pushy-branch: not found

./pushy-checkout b1

/bin/sh: 1: ./pushy-checkout: not found

sed -Ei 's/2/42/' 7.txt

cat 7.txt

1

42

3

4

5

6

7

./pushy-commit -a -m commit-2

/bin/sh: 1: ./pushy-commit: not found

./pushy-checkout master

/bin/sh: 1: ./pushy-checkout: not found

cat 7.txt

1

42

3

4

5

6

7

sed -Ei 's/5/24/' 7.txt

cat 7.txt

1

42

3

4

24

6

7

./pushy-commit -a -m commit-3

/bin/sh: 1: ./pushy-commit: not found

./pushy-merge b1 -m merge-message

/bin/sh: 1: ./pushy-merge: not found

cat 7.txt

1

42

3

4

24

6

7

Testing

Autotests

As usual, some autotests will be available:

2041 autotest pushy pushy-\*

...

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

2041 autotest pushy subset1 pushy-\*

...

2041 autotest pushy subset1\_13 pushy-\*

...

If you are using extra Shell files, include them on the autotest command line.

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.

You will need to do most of the testing yourself.

Test Scripts

You should submit ten Shell scripts, named test00.sh to test09.sh, which run pushy commands that test an aspect of Pushy.

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.

You are only expected to write test scripts testing parts of Pushy you have attempted to implement. For example, if you have not attempted subset 2 you are not expected to write test scripts testing pushy-merge .

Permitted Languages

Your programs must be written entirely in POSIX-compatible shell.

Your programs will be run with dash, in /bin/dash. You can assume anything that works with the version of /bin/dash on CSE systems is POSIX compatible.

Start your programs with:

#!/bin/dash

If you want to run these scripts on your own machine — for example, one running macOS — 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 systems, /bin/sh is the Bash shell: /bin/sh is a symlink to /bin/bash. Bash implements many non-POSIX extensions, including regular expressions and arrays. These will not work with /bin/dash, and you are not permitted to use these for the assignment.

You are not permitted to use Perl, Python or any language other than POSIX-compatible shell.

You are permitted to use only these external programs:

basename

bunzip2

bzcat

bzip2

cat

chmod

cmp

combine

cp

cpio

csplit

cut

date

dc

dd

df

diff

dirname

du

echo

egrep

env

expand

expr

false

fgrep

find

fold

getopt

getopts

grep

gunzip

gzip

head

hostname

ifne

less

ln

ls

lzcat

lzma

md5sum

mkdir

mktemp

more

mv

nl

patch

printf

pwd

readlink

realpath

rev

rm

rmdir

sed

seq

sha1sum

sha256sum

sha512sum

sleep

sort

sponge

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.

Note you are permitted to use built-in shell features including: cd, exit, for, if, read, shift and while.

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

You may submit extra shell files.