Week 05 Tutorial Sample Answers

1. The assignment specification doesn't fully explain the assignment - what can I do?

> ANSWER:
>
> A big part of the assignment is understanding what git does.
>
> You'll spend more time doing this than implementing subset 0.
>
> You can also use the the reference implementation **2041 tigger** to discover what your program is supposed to do in any situation.

2. How hard are the subsets?

> ANSWER:
>
> Once you understand what you have to dom subset 0 is not that hard.
>
> Subset 1 is a little harder.
>
> Subset 2 really hard.
>
> But note the marking scheme recognizes the difficulty of subsets 1 & 2.

3. What does **git init** do?

   How does this differ from **tigger-init**?

> ANSWER:
>
> **git init** creates an empty repository as does **tigger-init**
>
> **git init** uses the sub-directory **.git**
>
> **tigger-init** uses the sub-directory **.tigger**
>
> **git init** creates many files and sub-directories inside the directory **.git**
>
> **tigger-init** only *needs* to create the **.tigger** sub-directory.
> but *can* create other things inside **.tigger** sub-directory.

4. What do **git add *file*** and **tigger-add *file*** do?

> ANSWER:
>
> Adds a copy of ***file*** to the repository's **index**.
>
> It must be stored in the **.tigger** directory.
> But exactly how you store it is up to you.

5. What is the index in **tigger** (and **git**), and where does it get stored?

> ANSWER:
>
> Files get added to the repositoy via the index so its somethimes called a staging area.
>
> It must be stored in the **.tigger** directory.
> exactly how you store it is up to you.
>
> You might create a directory **.tigger/index/** and store the files there.

6. What is a commit in **tigger** (and **git**), and where does it get stored?

> ANSWER:

A commit preserves the state of all files in the index.

It must be stored in **.tigger**.
exactly how you store it is up to you.

You might create a directory **.tigger/*commit-number*/** and store the files there.

7. Apart from the **tigger-*** scripts what else do you need to submit (and give an example)?

ANSWER:

10 test files - **test00.sh** .. **test09.sh**

Here is an example of a suitable test file:

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

# ============================================================================
# test00.sh
# Test the tigger-add command.
#
# Written by: Dylan Brotherston <d.brotherston@unsw.edu.au>
# Date: 2022-06-20
# For COMP2041/9044 Assignment 1
# ============================================================================

# add the current directory to the PATH so scripts
# can still be executed from it after we cd

PATH="$PATH:$(pwd)"

# Create a temporary directory for the test.
test_dir="$(mktemp -d)"
cd "$test_dir" || exit 1

# Create some files to hold output.

expected_output="$(mktemp)"
actual_output="$(mktemp)"

# Remove the temporary directory when the test is done.

trap 'rm "$expected_output" "$actual_output" -rf "$test_dir"' INT HUP QUIT TERM EXIT

# Create tigger repository

cat > "$expected_output" <<EOF
Initialized empty tigger repository in .tigger
EOF

tigger-init > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

# Create a simple file.

echo "line 1" > a

# add a file to the repository staging area

cat > "$expected_output" <<EOF
EOF


tigger-add a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

# commit the file to the repository history

cat > "$expected_output" <<EOF
Committed as commit 0
EOF


tigger-commit -m 'first commit' > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
```

```
fi

# Update the file.

echo "line 2" >> a

# update the file in the repository staging area

cat > "$expected_output" <<EOF
EOF


tigger-add a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

# Update the file.

echo "line 3" >> a

# Check that the file that has been commited hasn't been updated

cat > "$expected_output" <<EOF
line 1
EOF

tigger-show 0:a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

# Check that the file that is in the staging area hasn't been updated

cat > "$expected_output" <<EOF
line 1
line 2
EOF


tigger-show :a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

# Check that invalid use of tigger-show give an error

cat > "$expected_output" <<EOF
tigger-show: error: invalid object a
EOF


tigger-show a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

cat > "$expected_output" <<EOF
tigger-show: error: unknown commit '2'
EOF

tigger-show 2:a > "$actual_output" 2>&1
```

```
if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

cat > "$expected_output" <<EOF
tigger-show: error: unknown commit 'hello'
EOF

tigger-show hello:a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

cat > "$expected_output" <<EOF
tigger-show: error: 'b' not found in commit 0
EOF

tigger-show 0:b > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

cat > "$expected_output" <<EOF
tigger-show: error: 'b' not found in index
EOF

tigger-show :b > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

cat > "$expected_output" <<EOF
usage: tigger-show <commit>:<filename>
EOF

tigger-show > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

cat > "$expected_output" <<EOF
usage: tigger-show <commit>:<filename>
EOF

tigger-show 0:a 0:a > "$actual_output" 2>&1

if ! diff "$expected_output" "$actual_output"; then
    echo "Failed test"
    exit 1
fi

echo "Passed test"
exit 0
```

8. You work on the assignment for a couple of hour tonight.
   What do you need to do when you are finished?

ANSWER:

Submit the latest version of your code with `give` .
Do this every time you work on the assignment.

Part of the assignment requirements is that you must submit intermediate versions of your code.

Additionally, submitting your code works as a backup.
You can view and retrive previous versions of your files from the autotest & submission page

9. Write a shell script *extract.sh* that, when given one or more archive files as command line arguments, will use the correct program to extract the files.

ANSWER:

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

case $# in
  0)
    echo "Usage: $0 <file> [<file> ...]"
    exit 2
    ;;
esac

status=0

for archive in "$@"; do
    if [ ! -f "$archive" ]; then
        echo "$0: error: '$archive' is not a file" >&2
        status=1
        continue
    fi

    case "$archive" in
        *.tar.bz2) tar xjf    "$archive" ;;
        *.tar.gz)  tar xzf    "$archive" ;;
        *.tar.xz)  tar xJf    "$archive" ;;
        *.bz2)     bunzip2    "$archive" ;;
        *.rar)     rar x      "$archive" ;;
        *.gz)      gunzip     "$archive" ;;
        *.tar)     tar xf     "$archive" ;;
        *.tbz2)    tar xjf    "$archive" ;;
        *.tgz)     tar xzf    "$archive" ;;
        *.zip)     unzip      "$archive" ;;
        *.jar)     unzip      "$archive" ;;
        *.Z)       uncompress "$archive" ;;
        *.7z)      7z x       "$archive" ;;
        *)
            echo "$0: error: '$archive' cannot be extracted" >&2
            status=1
        ;;
    esac
done

exit $status
```

10. Given an annonymous list of CSE logins.

    Write a shell script *last.sh* that, using shell `case` statments, finds the number of loggins that occurred from within UNSW. (Look for connections to from the uniwide network)

    Additionally, find the distribution of zIDs by their first digit.

ANSWER:

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

counter=0
z0=0
z1=0
z2=0
z3=0
z4=0
z5=0
z6=0
z7=0
z8=0
z9=0
class=0

while read -r loggin tty address other; do
    case "$address" in
        *uniwide*)
            counter=$((counter + 1))
            ;;
    esac
    case "$loggin" in
        z0*) z0=$((z0+ 1)) ;;
        z1*) z1=$((z1+ 1)) ;;
        z2*) z2=$((z2+ 1)) ;;
        z3*) z3=$((z3+ 1)) ;;
        z4*) z4=$((z4+ 1)) ;;
        z5*) z5=$((z5+ 1)) ;;
        z6*) z6=$((z6+ 1)) ;;
        z7*) z7=$((z7+ 1)) ;;
        z8*) z8=$((z8+ 1)) ;;
        z9*) z9=$((z9+ 1)) ;;
        *) class=$((class+ 1)) ;;
    esac
done < last.log

echo "$counter uniwide loggins"
echo "z0: $z0, z1: $z1, z2: $z2, z3: $z3, z4: $z4, z5: $z5, z6: $z6, z7: $z7, z8: $z8, z9: $z9"
echo "class: $class"
```

11. Write a shell function *top_and_bottom* that, given a file name, prints the file name, plus the first and last lines of the file.

```
$ . top-and-bottom.sh
$ top-and-bottom /usr/share/dict/british-english-insane
=================
/usr/share/dict/british-english-insane
-----------------
A
événements
=================
```

ANSWER:

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

top_and_bottom() {
    echo "================="
    echo "$1"
    echo "-----------------"
    sed -n '1p;$p' "$1"
    echo "================="
}

for file in "$@"; do
    top_and_bottom "$file"
done
```

12. Write a shell function *print_message* that, given an optional exit status and a message:

If no exit status is given the program should print a warning
If an exit status is given the program should print an error and exit the program

ANSWER:

```
#! /usr/bin/dash

print_message() {
    if [ $# -gt 1 ]; then
        echo "$0: error: $2"
        exit $1
    else
        echo "$0: warning: $1"
    fi
}

echo "This will print a warning"
print_message "This is a warning"

echo "This will print an error"
print_message 1 "This is an error"
```

13. Create a git repository called *cs2041-Labs* and add you week01 and week02 lab work.

    Then push your repository to the CSE gitlab servers.

ANSWER:

First, create the git repository:

```
$ git init cs2041-Labs
Initialized empty Git repository in /.../cs2041-Labs/.git/
$ cd cs2041-Labs
```

If applicable, set your default branch to name.

```
# `master` is the most common default branch name, but you can use any name
$ git config --global init.defaultBranch master
$ git branch -m master
```

Then copy accross your week01 and week02 lab work.

```
$ cp ../week01/*_answers.txt ../week02/*_answers.txt .
```

Next, add the files to the index.

```
$ git add .
```

And commit your work.

```
$ git commit -m "add Week01 and Week02 Lab work to git"
```

Now create an SSH key so that you can connect to the CSE gitlab servers.

```
$ mkdir -p ~/.ssh
$ cd ~/.ssh
$ ssh-keygen -t ed25519
# follow the instructions given
$ cat id_ed25519.pub
# copy the public key
# Don't show anyone your private key
$ eval `ssh-agent`
$ ssh-add id_ed25519
```

Add the key to the CSE gitlab servers.

Login with your zID and zPass. (make sure you have the "UNSW" tab selected)

In the top right corner, click on the "Preferences" button.

On the left hand side, click on the "SSH Keys" tab.

Paste the public key in the "Key" field.
Add a desciptive title so you know what computer the key is for.

(This key has already been deleted and won't do anything)

Create a new project, in the top middle of the page.

Create a blank project.

Add a project name.
And make sure that the "Project URL" is set to your zID.
Then add a description.
And make sure that the project is set to **private**.
And make sure that you **do not** add a README

If you have not done so before, follow the instructions to configure your git information.

```
$ git config --global user.name "Dylan Brotherston"
$ git config --global user.email "d.brotherston@unsw.edu.au"
```

Follow the instructions to "Push an existing Git repository".

```
$ git remote add origin gitlab@gitlab.cse.unsw.EDU.AU:z5115658/cs2041-labs.git
$ git push -u origin master
# `master` should be whatever your default branch name is
```

Refresh the GitLab page and you should see your files.

14. There is a git repository located on the CSE gitlab servers at https://gitlab.cse.unsw.edu.au/cs2041/22t2-tut05
    Clone this repository to your local machine.

ANSWER:

On gitlab find the URL of the repository.

Once you have copied the URL, you can clone the repository.

```
$ git clone gitlab@gitlab.cse.unsw.EDU.AU:cs2041/22t2-tut05.git
```
