Cheat Sheet

This page contains some hints and tips for carrying out the "Bug Hunting with Git" activity. Git actually contains a range of techniques for helping to track down bugs in code. In this activity, we just look at how we use Git to record a bug fix, as a separate commit, rather than any more advanced features.

If you need more help, you can always post a question on the activity itself, on Future Learn.

Step 1: Acquire a copy of the code base

You'll need your own remote version of this repository to complete the activity, as well as creating your own local clone.

First you need to decide which hosted Git system you're going to create your remote repository on. The easiest approach is to create a GitLab.com account and fork the repository there. But you can host your remote on GitHub, BitBucket or any other Git hosting site you can get access to.

Help in carrying out this step is available at the links below, should you need it. Choose either one of these options to carry out the activity.

- Creating a Remote Repository (Fork) on GitLab.com
- Creating a Remote Repository on Other Git Hosting Services

Step 2: Create a Feature Branch to Work on

First choose the name for your branch. Then checkout the commit you want to use as the starting point for the branch, and create the new branch at that point. Make sure this new branch is checked out before you start work on the bug fix.

The Git commands to use to carry out this step are:

```
git switch main

git branch <bug-fix-branch>

git switch <bug-fix-branch>
```

where `<bug-fix-branch>` is the name of your new branch. A suitable name in this case might be `multi-solution-bugfix`, for example.

You can also use the `-c` flag to create and switch to a branch in a single instruction:

```
git switch -c multi-solution-bugfix
```

Step 3: Find and Fix the Bug on the Branch

Follow the links below to get information on how to run the tests, and see how the bug causes some of them to fail. This information can help you work out what the bug is and how to fix it.

Running the Simploku Tests

If you just want practice at using Git to record the details of the bug fix, and including them into the main development line of your project, here is information about the line that contains the bug and what you need to change to fix it.

The Simploku Bug

The Git commands needed to commit the bug fix to your local repository once you have made it are:

```
git add simploku.py

git commit -m 'Fix bug in solver for puzzles with several solutions'
```

At this point, your Git graph should look like this:

```
git log --all --decorate --oneline --graph



* b688133 (HEAD -> multi-solution-bugfix) Fix handling of puzzles with multiple solutions

* 5b05abd (main) Initial import
```

Step 4: Push the Commit to the Remote Repository

The Git command to complete this step is:

```
git push origin
```

Step 5: Merge the code into the Mainline

In a full professional workflow, we would check that the tests run and that the code passes all quality checks and code review at this stage. We would also fetch down and integrate any changes to the Git repository that our team mates may have made since we last synchronised.

The Git commands needed for this step of the activity are as follows. First, check out the branch we want the new code changes to be included into:

```
git switch main
```

Next, we make the merge:

```
git merge <bug-fix-branch>
```

where `<bug-fix-branch>` is replaced with the name of the branch you created to hold your bug fix.

If this has gone well, you should get the message:

```
Updating 5b05abd..b688133

Fast-forward

simploku.py | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)
```

(Of course, the commit SHAs will be different for your merge.)

Your Git graph should now look like this:

```
git log --all --decorate --oneline --graph



* b688133 (HEAD -> main, multi-solution-bugfix) Fix handling of puzzles with multiple
solutions

* 5b05abd Initial import
```

Step 6: Share the New Mainline with your Team Mates

The Git command for this step is:

```
git push origin
```

Creating A Copy of the Repository on Other Git Hosting Services

If you prefer to host your remote repository for this exercise on a Git hosting service other than GitLab, you can follow the instructions here. It's a bit more work than creating a fork on the same hosting service as the repository you want to copy, but isn't too much trouble.

To fork on GitLab.com, we create our own personal copy of the code as a remote repository first, and then clone it. To move the code from GitLab.com to GitHub.com or some other hosting site, we're going to follow this same

process but the other way around. We're going to create our own personal clone from the GitLab.com project, and then push to that to a remote repository on our preferred Git hosting site.

So, the first step is to clone this project on GitLab.com. You can use the HTTPS version of the project URI for this, since it is a publicly accessible project and no authentication is needed:

```
git clone git@gitlab.com:collaborative-coding-with-git/bug-hunting-with-git.git
```

If we look at the remote repository set up for this, we'll see that the original GitLab repo is set as `origin`.

```
$ git remote -v

origin  git@gitlab.com:collaborative-coding-with-git/bug-hunting-with-git.git (fetch)

origin  git@gitlab.com:collaborative-coding-with-git/bug-hunting-with-git.git (push)
```

We next need to remove this link between your personal clone and the main GitLab project. We can do this with the following command:

```
git remote remove origin
```

Next, we'll set up a new remote project, in your preferred hosting site.

Log in to your preferred Git hosting site, and create a new project. You can call the project whatever you like. It doesn't need to be the same as the name of the GitLab project you'll be copying from, though it will probably be easier to keep track of the purpose of the project if you give it a similar name.

The next step is to copy the URI for this new project to our clipboard, as if we were planning to clone the project. Instead, we're going to set the project as the new remote repository for our local repository. Use the following command for this:

```
git remote add origin <uri>
```

where `<uri>` is the clone URI for the project you just created on your preferred Git hosting platform. You can check that this has worked using the command:

```
git remote -v
```

once again. You should see a remote called `origin`, pointing to your preferred new project for both push and fetch operations.

The final step is to push the code to the new remote. You can use the following command to do this:

```
git push --all origin
```