Task 1A - Using Git

100 % 13 of 13 items complete

Configuring Git

1) Steps:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

Setting up a Local Git Repository

2) Steps:

- Open a terminal window.
- Create a folder in your home directory.

```
mkdir Project01
```

- Change into that directory.

```
cd Project01
```

- Use a text editor to create a couple of text files, called **Doc01.txt** and **Doc02.txt**.

  Although it doesn't matter what text we put in the files for this exercise, we may as well try and learn a bit of Kotlin while we are at it. Put the following text into each file:

  Doc01.txt:

```
// Make the function start return "Hello World"
fun start(): String = TODO()
```

New Folder

```
// This function takes two arguments.  Their default values are
fun checkAge(age:int=0, year: int=2021): boolean
```

- Now initialise a Git repository in that directory, by typing the following at the command line:

```
git init
```

- This will create a new subdirectory named .git which will contain all of the file change information for all versions of the project files. This subdirectory may be hidden, but it is there.

Committing the changes

☐ 3) Steps:

- At this point, none of the files in the project directory are being tracked. None of them are under version control.
- To place individual files under version control, do the following:

```
git add Doc01.txt
git add Doc02.txt
```

- Or, if you want to do them all at once:

```
git add Doc*.*
```

- Even more quickly, if you want to track all of the files in the current directory

```
git add ––all
```

although you often will not want to track absolutely every file, so be careful when using this option.
- At this point you have told GIT what changes to record (you **add**ed them to the change set).
- Now you need to write the change information for all the tracked files into the repository (with some message so you know what it is about later):

```
git commit –m "This is the first version of the project"
```

New Folder

Current state of the working directory

☐ 4) Steps:

- To find out what the current state of the repository is, type:

```
git status
```

- it should display the following message:

```
On branch master
nothing to commit, working tree clean
```

- So lets make some changes: edit `Doc01.txt` and create a new file called `Doc03.txt`, as shown below.
  `Doc01.txt`

```
// Make the function start return "Hello World"
fun start(): String = "Hello World"
```

`Doc03.txt`

```
val numbers = listOf(1, -2, 3, -4, 5, -6)
val totalCount = numbers.count()
```

- Now type:

```
git status
```

- You should get a message saying something like:

```
On branch master

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working

        modified: Doc01.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Doc03.txt
```

New Folder

```
no changes added to commit (use "git add" and/or "git commit -a"
```

Reading this takes a bit of getting used to but it means:
  - `Doc01.txt` was changed, but not added to the changeset for recording yet.
  - `Doc03.txt` does not exist in the repository and wasn't added to the changeset either
  - Nothing was actually added to the changeset, so you can't commit as there is nothing to commit.
- As `Doc01.txt` is being tracked,the new changes need to be staged. Do so:

```
git add Doc01.txt
```

- As `Doc03.txt` is not yet being tracked, the whole file needs to be staged. Do so:

```
git add Doc03.txt
```

- Now type git status again.

```
git status
```

- You should see that the changes are now staged, and ready to be committed.
- To commit, type the following:

```
git commit -m "First set of changes"
```

- If you just type git commit without specifying a message (without `-m "message"`), Git will open your default text editor and wait for you to type your message there. After you close the editor (and wrote any message) the commit will happen.

Viewing the history of changes

☐ 5) steps:

- If you want a list of all the commits you have made, type:

```
git log
```

- and you will get something like this:

```
commit 66563cd9d1bd90d0685fd5ff1a5d340e5a17ded1
Author: kwilson <kwilson@bournemouth.ac.uk>
Date: Sun Jul 7 19:32:04 2019 +0100
```

New Folder

```
commit 89b8f5cce3fea06d635decd514711768202679e6
Author: kwilson <kwilson@bournemouth.ac.uk>
Date: Sun Jul 7 19:12:55 2019 +0100

    This is the first version of the project
```

- Note how each commit is identified by a long hash code, the author id/email, a timestamp, and the commit message. The commit message is there to remind you where you had got up to in the development process, when you committed that particular version.

Branching

6) Steps:

Sometimes, you want to make a copy of your project, to try out some changes, without messing up the original. Git allows you to branch your project, and commit changes to the branch. Once you have tested your ideas, you can then switch back to the main branch, where the original unaltered project is still stored, and then carry on developing that.

The default branch name in Git is **`master`** although there is nothing special about that name. It can be called whatever you want.

You can create a new branch called '**`testing`**' by typing:

```
git branch testing
```

WARNING!

The branch command only created a new branch – it didn't switch to it.

The branch starts at the current point in the history (normally the head/last change)

Switching Branches

New Folder

- Git knows which branch you are currently on by using a pointer called '`HEAD`'. If you type:

```
git log --oneline --decorate
```

- You will get something like:

```
f30ab (HEAD -> master, testing) add feature #32
34ac2 Fixed bug #1328 - stack overflow under certain conditions
98ca9 The initial commit of my project
```

- which indicates that the current branch is '`master`'.

- You can switch branch by using checkout. Type this:

```
git checkout testing
```

- Now if you type `git log --oneline --decorate` note that HEAD is pointing to the '**`testing`**' branch.

- We will now make some changes so that the branches are different. Do the following:

- Open **`Doc01.txt`** and add another line so that it now contains:
  `Doc01.txt`

```
// Make the function start return "Hello World"
fun start(): String = "Hello World"
// This is the testing branch version of doc01.txt
```

- Delete `Doc02.txt`
- Create another file called `Doc04.txt` as shown below:
  `Doc04.txt`

```
// Note the custom setter - the counter property is incremented
// the variable is assigned a value.

class PropertyExample() {
    var counter = 0

    var propertyWithCounter: Int? = null
        set(v: Int?) {
            field = v
            counter++
        }
}
```

- Do git status, to see what the current state of your working directory is,

New Folder

- and stage the changes by typing:

```
git add Doc01.txt
git add Doc02.txt
git add Doc04.txt
```

- **Note**. Although `Doc02.txt` has been deleted, what you are adding to the changeset is that change - the fact that the file is no longer present.
- Commit the changes by typing:

```
git commit -m "First commit in the testing branch"
```

7.2)

- Change the contents of `Doc01.txt` and `Doc03.txt`
  - `Doc01.txt`

```
// Make the function start return "Hello World"
fun start(): String = "Hello World"
// This is the testing branch version of doc01.txt
// This is still the testing branch of doc01.txt
```

  - `Doc03.txt`

```
val numbers = listOf(1, -2, 3, -4, 5, -6)
val totalCount = numbers.count()
val evenCount = numbers.count { it % 2 == 0 }
```

- Stage and commit the changes.

7.3)

- Before you switch back to the master branch, get a file manager open so that you can see the contents of the Project01 folder when you carry out the next command.
- Now switch back to the original branch.

```
git checkout master
```

- You should be able to see the files change.
- List the contents of your project folder just to double check :

New Folder

`dir` (on Windows)

- Note carefully what has happened.
- **THE CHECKOUT COMMAND WILL CHANGE THE CONTENTS OF YOUR WORKING DIRECTORY**.

  The file that you deleted has come back. The file that you added isn't there. Check the contents of `Doc01.txt`. The extra line you added has gone.

  Your project has reverted back to the state it was in before you branched it.

  ALWAYS BE AWARE OF WHICH BRANCH YOU ARE CURRENTLY ON, WHEN DEVELOPING A PROJECT.

  Use the git status command to remind you. And always commit all changes before you do anything "dangerous".

7.4)

1. Make the following changes to the master branch. Add a file called **Doc05.txt** and modify the contents of **Doc01.txt.** as shown below:

   `Doc01.txt`

   ```
   // Make the function start return "Hello World"
   fun start(): String = "Hello World"
   // This file is in the master branch
   ```

   `Doc05.txt`

   ```
   // All examples create a function object that performs upper-cas
   val upperCase1: (String) -> String = { str: String -> str.toUppe
   // A lambda in all its glory, with explicit types everywhere.
   // The lambda is the part in curly braces
   ```

2. Stage and commit the changes.
3. Now switch back to the `testing` branch, using `git checkout testing` keeping your eye on the contents of the `Project01` folder in the file manager. Again, you should see the change.
4. Use `git status` to make sure that you are on the right branch.
5. List the files – you should now see the files that were there, the last time you were on the `testing` branch. The contents of the working directory have changed again.

New Folder

6. Make some more changes, stage and commit them.

7. Go back to the master branch.

8. Add the following lines to **`Doc05.txt`**.

   `Doc05.txt`

```
// All examples create a function object that performs upper-cas

val upperCase1: (String) -> String = { str: String -> str.toUppe

// A lambda in all its glory, with explicit types everywhere.
// The lambda is the part in curly braces

val upperCase5: (String) -> String = { it.toUpperCase() }

// For lambdas with a single parameter, you can use the implicit
```

9. Stage and commit the changes.

Show the Commit Graph

☐ 8) Steps:

If you want to see the commit history in a graphical form, type:

```
git log --graph --decorate --all --color
```

Going back to a previous version

☐ 9) Steps:

- If you want to go back to an earlier version of the project, you can refer to it by using it's hash code – the 8 character alphanumeric string displayed before the commit message. (It's actually the first 8 chars of a 40 character string. In the following command, replace the hash code **`09345def`** with the one that your listing shows for

New Folder

h. The one with the commit message "xxxxx".

```
git checkout -b oldversion01 09345def
```

- What this will do is create a new branch called '**`oldversion01`**' at that commit point, so that you can start making changes starting with the files as they were at that commit. You have effectively rolled back the files in your working directory to how they were at that point. You can then start making changes on the new branch to take development in a new direction.
- Make some more changes to the Doc files, stage and commit.
- Show the commit graph again to see the changes.

Using Graphical Clients

☐ 10) Steps:

1. There are many graphical clients for Git, with different features. One of the most straightforward is the one that comes packaged with the standard git installation. Type:

```
git gui
```

2. And you will see the basic screen.
3. Then go to **Repository > Visualize All Branch History** and you will see a graphical commit tree.
4. If you highlight a particular commit, it will show you the line by line changes that were made to each file.
5. I find that it is quicker to do the basic status/add/commit commands from the command line, but the gui is better for visualising the whole tree and working out where you did something.
6. Just to get you started with the gui, go back to the basic screen, and then use the text editor to change a couple of the Doc text files.
7. Then click the rescan button towards the bottom of the basic gui screen.
8. In the unstaged changes pane, click on the little blue document icons to stage them.
9. Type a commit message, then click on the Commit button.
10. If you go back to the Visualise All Branch History window, and do File > Reload, you will see the new commit.
11. Btw. there is also a graphical client built in to Android Studio/Intellij

New Folder

Other resources

No Items

Activity Details

Well done! You have completed the checklist

Last Visited 27 February, 2023 11:54 AM