Git Branch

Working with Git Branches
In Git, a `branch` is a new/separate version of the main repository.

Let's say you have a large project, and you need to update the design on it.

How would that work without and with Git:

Without Git:

- Make copies of all the relevant files to avoid impacting the live version
- Start working with the design and find that code depend on code in other files, that also need to be changed!
- Make copies of the dependant files as well. Making sure that every file dependency references the correct file name
- EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
- Save all your files, making a note of the names of the copies you were working on
- Work on the unrelated error and update the code to fix it
- Go back to the design, and finish the work there
- Copy the code or rename the files, so the updated design is on the live version
- (2 weeks later, you realize that the unrelated error was not fixed in the new design version because you copied the files before the fix)

With Git:

- With a new branch called new-design, edit the code directly without impacting the main branch
- EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
- Create a new branch from the main project called small-error-fix
- Fix the unrelated error and merge the small-error-fix branch with the main branch
- You go back to the new-design branch, and finish the work there
- Merge the new-design branch with main (getting alerted to the small error fix that you were missing)

Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

Branching in Git is very lightweight and fast!

New Git Branch

Let add some new features to our `index.html` page.

We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new `branch`:

```
=> git branch hello-world-images
```

Now we created a new `branch` called "`hello-world-images`"

Let's confirm that we have created a new `branch`:

```
=> git branch

  hello-world-images

* master
```

We can see the new branch with the name "hello-world-images", but the `*` beside `master` specifies that we are currently on that `branch`.

`checkout` is the command used to check out a `branch`. Moving us from the current `branch`, to the one specified at the end of the command:

```
=> git checkout hello-world-images

Switched to branch 'hello-world-images'
```

Now we have moved our current workspace from the master branch, to the new `branch`

We have made changes to a file and added a new file in the working directory (same directory as the `main branch`).

Now check the status of the current `branch`:

Using `--all` instead of individual filenames will Stage all changed (new, modified, and deleted) files.

Check the `status` of the `branch`:

We are happy with our changes. So we will commit them to the `branch`:

Now we have a new `branch`, that is different from the master `branch`.

Note: Using the `-b` option on `checkout` will create a new branch, and move to it, if it does not exist

Switching Between Branches

Now let's see just how quick and easy it is to work with different branches, and how well it works.

We are currently on the branch `hello-world-images`. We added an image to this branch, so let's list the files in the current directory:

```
 ls
```

'git_tuto 2.pdf' 'git_tuto 4.pdf' 'git_tuto 6.pdf'

'git_tuto 3.pdf' 'git_tuto 5.pdf'   index.html

We can see the new file `img_hello_world.jpg`, and if we open the html file, we can see the code has been altered. All is as it should be.

Now, let's see what happens when we change branch to `master`

`img_hello_world.jpg` is no longer there! And if we open the html file, we can see the code reverted to what it was before the alteration.

See how easy it is to work with branches? And how this allows you to work on different things?

Emergency Branch

Now imagine that we are not yet done with hello-world-images, but we need to fix an error on master.

I don't want to mess with master directly, and I do not want to mess with hello-world-images, since it is not done yet.

So we create a new branch to deal with the emergency:

Now we have created a new branch from master, and changed to it. We can safely fix the error without disturbing the other branches.

Let's fix our imaginary error:

We have made changes in this file, and we need to get those changes to the master branch.

Check the status:

stage the file, and commit:

Now we have a fix ready for master, and we need to merge the two branches.