* Each project that you make in GitHub can have its own website
  + GitHub doesn’t allow for private projects under free plan
* Bitbucket only allows you to create 1 website per account
  + allows you to create as many private projects as you'd like
* Markdown and the README.md file
  + # = largest heading
  + ## = subheading
  + \_\_bold\_\_
  + \_italics\_
  + `code` and ```code blocks```
  + > quotes
* .gitignore file
  + write which files you want to ignore
  + just write which files you want to ignore on each line
* The Structure of a Repository
  + repository: where we store the files and their versions (it has a couple of layers)
  + Layers in a repository:

|  |
| --- |
| Files pushed to cloud |
| Files committed to local repository |
| Files added to next commit  (STAGING AREA) |
| Files in your system |

push

commit

add

**Basics of Git**

* to add .git folder for repository **git init**
* What is a diff?
  + the *differences between one file and another* is a diff
  + let's say we have a file1.txt and an identical file2.txt
  + if we modify file2 and then create a diff of file1.txt and file2.txt we get something like this:

**$: diff file1.txt file2.txt -u**

--- file1.txt 2021-06-07 10:06:38.000000000 -0500

+++ file2.txt 2021-06-07 10:06:52.000000000 -0500

@@ -1,2 +1,2 @@

Hello, world!

-This is a test.

+We are testing!

* when we add files to the staging area, the files will be used to create diffs when we commit
* we add files to the staging area with **git add <file>**
* Adding files to the staging area
  + after running **git add** you can run **git status** to check changes to the working tree
  + you can run **git rm --cached <file>** to remove a file from the staging area
* Committing files to the local repository: git commit
  + you can commit **one file** (which adds one diff to the repository), or **many files** (which adds one diff per file)
    - *all the diffs you add at the same time* is one **commit**
  + when we have added all the files, we can commit them
  + this creates the diffs for each and stores them in the .git folder (do this by **git commit**)
* Commit Messages
  + we should add commit messages every time we commit
  + commit message with command line:
    - **git commit -m "<message>"**
  + commit message with text editor:
    - **git commit**
    - a text editor will pop up in the terminal where we can input our message
* Adding a remote to your repository
  + a remote is a copy of your Git repository that isn’t local to your machine
    - other people can access this repository if you give them permission
  + you can have many remotes for your repository
  + we need to add a remote before being able to use it
  + the command to add a remote is **git remote add**
    - **git remote add origin <bitbucket/github\_url>**
    - origin = new remote name
  + before creating a remote you will need to create a project on bitbucket or Github so you can have your URL
* Pushing and pulling to and from remote repository
  + Pulling changes
    - as long as you are working with **only one branch**, you can use git pull to bring changes from the cloud to your local repository
    - this gets the *new commits* and merges them with our local repository
  + Fetching changes
    - a fetch is the same as a pull, but it doesn’t merge the changes into our local repository
    - so*, it just gets the commits from the cloud and stores them in the .git folder*
    - you can merge all of them, or some of them
  + Problem with pull
    - the problem with pull comes when you are using multiple branches
    - it is better to use **git fetch** and **git merge** than it is to use git pull
  + PUSHING and PULLING
    - git add origin <URL>
    - git merge origin/master
    - git commit -m "*commit message*"
    - git push origin
      * if error encountered run git push --set-upstream origin master
      * git merge origin/master
    - git push origin

**Branches and Workflows**

* What is a branch?
  + Workflow described
    - initially, we have no commits, and nothing added to our local repository
    - then we add some files
    - then we commit them
    - somewhere inside the .git folder, the commit gets stored
  + A new Commit
    - if we make changes, and add them, and commit them, we don’t get a new file
    - the new commit is *also added to the same file*
    - so now the file that stored the original commit now references **two different commits** (and their details)
    - when we make our first commit,

it gets stored (C0)

* + - the next commit stored below

C0 is what we call C1

* + - then C2, C3, etc…
  + A Branch
    - a Branch is a bunch of commits
      * the last commit and the commits above it
    - a branch is a file that contains the different commits (previous example)
      * branch is used to store commits
    - branches have names (master, feature 1, etc…)
* Creating a Branch
  + we always have on branch, master, to begin with
  + we can create new branches using **git branch <*branch name*>**
  + git checkout allows you to move between branches
    - **git checkout <branch\_name>**
    - you can checkout to any branch
    - you can also checkout to any commit
  + git checkout master~2
    - like checking out on commit master[2]
* Creating branches (terminal)
  + Scenario: going to add a new feature to our program, but so we don’t disturb others working in the master branch, we a create a new branch so we can make our changes there

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git branch nicksBranch

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git branch

branch

\* master

nicksBranch

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git checkout nicksBranch

Switched to branch 'nicksBranch'

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git branch

branch

master

\* nicksBranch

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ ls

total 16

-rw-r--r-- 1 nicholausbrell staff 175 Jun 7 11:02 README.md

-rwxr-xr-x 1 nicholausbrell staff 56 Jun 7 10:59 **main.py**

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ vi main.py

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git add main.py

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git status

On branch nicksBranch

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: main.py

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git commit -m "added a user input feature to main.py"

[nicksBranch d48fa22] added a user input feature to main.py

Committer: Nicholaus Brell <nicholausbrell@nicholauss-mbp.utsarr.net>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 3 insertions(+)

nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ git push origin nicksBranch

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 8 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 369 bytes | 369.00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

remote:

remote: Create a pull request for 'nicksBranch' on GitHub by visiting:

remote: https://github.com/nickb210/Udemy-Repository/pull/new/nicksBranch

remote:

To https://github.com/nickb210/Udemy-Repository

\* [new branch] nicksBranch -> nicksBranch

* Bringing in Branches from origin
  + say we created a new branch *feature-1* on our remote repository
  + nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository**$ git fetch**

From <https://github.com/nickb210/Udemy-Repository>

\* [new branch] feature-1 -> origin/feature-1

* nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ **git checkout -b feature-1 origin/feature-1**

Branch 'feature-1' set up to track remote branch 'feature-1' from 'origin'.

Switched to a new branch 'feature-1'

* + this creates a local branch and track origin/feature-1
* nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ **git branch**

branch

\* feature-1

master

nicksBranch

* nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ **git add main.py**
* nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ **git commit -m "Added a print statement in main.py for feature-1 branch"**

[feature-1 1d56364] Added a print statement in main.py for feature-1 branch

Committer: Nicholaus Brell <nicholausbrell@nicholauss-mbp.utsarr.net>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 1 insertion(+)

create mode 100644 main.py

* nicholausbrell@nicholauss-mbp:~/Desktop/Critical Design/Modules/CDQ2-46 Git By Example/Udemy-Repository$ **git push**

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 8 threads

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 352 bytes | 352.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/nickb210/Udemy-Repository

956d596..1d56364 feature-1 -> feature-1

* Deleting local and remote branches
  + to delete a local and remote branch

**git checkout master**

**git branch -d feature-1**

**git push origin :feature-1**

* Merging branches
  + in order to merge, *we need to be in the branch that is going to receive the changes*
  + then we use **git merge <branch>**
  + you need to be in parent branch to merge. You cannot be in the child branch and merge with its parent branch
  + True merge:
    - if two files are different, the changes of one are applied to the other if possible
  + Merge Error:
    - happens if the two branches make different changes on the same line
* Reverting changes
  + Possible Scenarios
    - **Before** you have pushed changes to the remote repo
    - **After** you have pushed the changes (someone might have done some work on these changes)
  + Before Pushing
    - you can do whatever you want
    - unless you are pushing to a branch in which only you are working, **never push things that will break the program**
    - keep these locally until you need to push them or they are complete to some extent
    - before pushing you are allowed to permanently delete commits
    - you can delete the commits as if they had never happened
      * **Warning: you cannot bring them back!**
  + Deleting commits
    - **git reset <commit>**
    - this will delete all commits up to and including the referenced <commit>
      * remember relative referencing of commits (HEAD^, HEAD~2)
  + After Pushing
    - someone might have worked with the code you pushed to the remote
    - you **shouldn’t delete commits irreversibly** just in case someone used them
    - what we can do is **revert** the commits, which creates a new commit that indoes what the previous did
    - Why revert?
      * if someone was using our commit, we can always re-revert the commit we reverted and bring it back like nothing changes
      * since nothing is deleted, there is always a chance to fix things if something went wrong
    - **git reset --hard HEAD^**
      * revert last commit to the previous commit
* What is the "Gitflow" workflow?
  + why is a workflow important?
    - prevents merge conflicts
    - cleans up history
    - makes sure everyone on team follows the same protocol
  + The Gitflow workflow
    - each person/group works in a branch
    - each branch is used for one thing: a new feature, bug fix, etc…
    - after work is finished in a branch, it is added to the main branch

**Gitflow and SourceTree**