Introduction to version control with Git

Install Git

I. Download: https://git-scm.com/downloads

II. Documentation: https://git-scm.com/doc

Local repository and local workflow

I. Create the repository

   A. Select the repo's root directory on the file system

   B. Initialise the repository: `% git init`

   C. We now have:

      1. a workspace, a.k.a. working copy

      2. an index, a.k.a. staging area

      3. a repository with a single branch, named *main* by default

II. Add a file to the repo

   A. Create the file at the root directory (e.g. `outline.md`)

   B. The newly created file is not yet tracked. This means that although it's in the workspace, it's not yet subject to versioning. We can see this by using the command `% git status`

   C. Add the file to the index: `% git add outline.md`

   D. Lets remove it from the index and add it again, just for show:
   ```
   % git rm —cached outline.md
   % git add outline.md
   ```

   E. Commit the changes to the repo: `% git commit -m “The first commit”`

   F. Check the result: `% git status`

III. Modify and existing file and create a new one

   A. Make the changes by editing `outline.md` and copying `github-git-cheat-sheet.pdf` to the folder

   B. Check the result: `% git status`

   C. Lets add the changes to the index and commit them in one single command:
   ```
   % git commit -a -m "The second commit”
   ```

   D. Check the result: `% git status`
   Notice that the newly created file was not included in the commit. Because its a new file, we

need to add it explicitly to the index, like so:
`% git commit -a -m "The second commit”`

E. We wanted this file to be included in the second commit, but we forgot to add it. That’s ok. We can fix the last commit by amending it:
`% git commit --amend`

F. Check the result and the commit history:
```
% git status
% git log
% git shortlog
```

G. Now we can *time travel* between commits =), using: `% git checkout <commit>`

H. Finally lets get back to the tip of our branch and leave detached mode, like this:
`% git checkout main`

Local repository with remote repository (solo dev workflow)

I. Create the remote repository, for example on GitHub

II. Add the remote repository as a remote for our local repo:
`% git remote add origin <remote_repo_url>`

III. Check the result with: `% git remote`

IV. Lets send the local repo’s contents to the remote repository: `% git push -u origin main`
Make sure you have a created a Personal Access Token. See here: https://docs.github.com/en/enterprise-server@3.4/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

---

Multiple devs workflow

I. Create a local repository by cloning the remote at Github
`$ git clone <remote_repo_url>`

II. Check the result with:
```
% git status
% git log
```

III. Lets modify a file on the newly created local repo (i.e. the clone) and commit the changes:
`% git commit -am "The third commit”`

IV. Check the result

V. Publish the new commit to the remote
`% git push`

VI. Check the result

VII. Gets the changes from the remote to the local repository, that is, the repository of the dev that originally created it: `% git pull` and check the result

VIII. Thus far we presumed that each dev is working in a distinct set of files, which is an ideal scenario. What if they end up changing the same file?

IX. Change file in one of the repos.

X. Commit to the local repo and publish to the remote and check the result:

```
% git commit -am "Concurrent commit 1"
% git push
% git status
```

XI. Change the file in the other repo

XII. Fetch changes from the remote, without committing them to the local repo. Check the results.

XIII. Commit to the local repo without pulling the changes from the remote and TRY to publish the changes made locally to the remote

```
% git commit -am "Concurrent commit 2"
% git push
```

XIV. A conflict is detected. Let's fix it. First, we pull the remote changes and merge them to the local repository

```
% git pull —ff
```

XV. Then, we resolve the merge conflicts, manually, if needed

XVI. Finally, we commit the changes and publish them to the remote:

```
% git commit -am "Concurrent commit 2"
% git push
```

XVII. Et voilá =)