

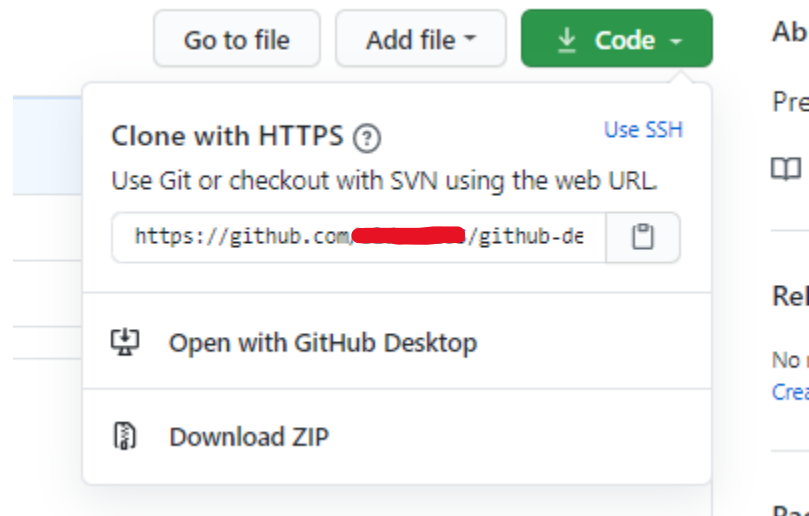
Git & Github walkthrough

Pre-requisites

1. Sign-up to Github site
2. Google “Download Git client” and install Git client on your workstation. The default options should be good enough to start with, you may google them after you better understand Git to fine tune its behavior.

Walkthrough steps

1. Create Github repo (public one) choose MIT license:
<https://docs.github.com/en/github/getting-started-with-github/create-a-repo>
 - a. A *Git repo* is a stand-alone container of files for which changes history and versions will be saved by Git for as long as the repo exists. Almost all Git commands are executed in the context of a repo – the top level unit of work.
 - b. MIT license means this repo is open source everyone can leverage, it’s the most popular open source license.
2. Clone the repo using your Git client
 - a. Take the repo URL from Github:



- b. In your cmd/bash session navigate to local folder where you want to place the repo and clone using `git clone <URL>`, e.g. `git clone https://github.com/[redacted]/github-demo-repo.git`

```
C:\Users\user> cd C:\Users\user\Documents\GitHub\Repos> git clone https://github.com/username/github-demo-repo.git
Cloning into 'github-demo-repo'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 1.26 KiB | 76.00 KiB/s, done.
```

3. Navigate to repo folder that was just cloned, e.g. `cd github-demo-repo`
 - a. The repo copy in Github is called the *remote repo*, and your local copy is called the *local repo*. Git is distributed source/version control system, where every computer can act as remote repo server for your local repo, however there's usually a single server all clients sync with, called *origin*, e.g. Github is the origin.
4. Check your repo status using `git status`

```
C:\Users\user\Documents\Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

- a. This output means the local repo branch called “master” reflects the state of remote repo “master” branch. Branches will be explained soon, for now it means we didn’t make a change to the repo’s content in a way that is not aligned with remote repo.
5. Check your commits history using `git log --graph --pretty=format:"%h %s"`

```
C:\Users\tezzenejo\Documents\git> cd C:\Users\tezzenejo\Documents\Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* 69d6f61 Initial commit
```

- a. A *Git commit* captures a change to a set of files in the repo. A Git repo starts with an initial commit, which captures the initial files and their contents, and then each change that's persisted to the repo is captured by another commit.
6. Create a file using *echo "file A" >> A.txt* and check status

```
C:\Users\k1337\Documents\Presentations\Github\Repos\github-demo-repo>echo "file A" >> a.txt
C:\Users\k1337\Documents\Presentations\Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

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

nothing added to commit but untracked files present (use "git add" to track)
```

- a. With Git a file in local repo can be in 3 states: committed, tracked, untracked. A new file starts as untracked, which means it's not part of the Git repo state.

7. To delete all local files that are untracked, use *git clean -xfd*

```
C:\Users\k1337\Documents\Presentations\Github\Repos\github-demo-repo>git clean -xfd
Removing a.txt
C:\Users\k1337\Documents\Presentations\Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

8. Create the file again, then run *git add -A* to make all untracked files tracked, then run *git clean -xfd* and observe the file wasn't deleted since it's now tracked

```

C:\Users\user\Documents\Github\Repos\github-demo-repo>echo "file A" >> a.txt
C:\Users\user\Documents\Github\Repos\github-demo-repo>git add -A
C:\Users\user\Documents\Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   a.txt

C:\Users\user\Documents\Github\Repos\github-demo-repo>git clean -xfd
C:\Users\user\Documents\Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   a.txt

```

- a. The term *staged* files is same as *tracked* files.
 - b. You can see the command *git restore* can be used to make files untracked again, I never use it.
9. Change the existing README.md file as well Github added in the initial commit, use `echo "README.md change 1" >> README.md`, then check status and witness the tracked new file and the changed committed file with unstaged change:

```

C:\Users\user\Documents\Github\Repos\github-demo-repo>echo "README.md change 1" >> README.md
C:\Users\user\Documents\Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   a.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

```

10. To regret uncommitted changes, whether it's the creation of new file that's tracked, or a change to already committed files, use `git reset --hard`:

```

C:\Users\...> \Github\Repos\github-demo-repo>git reset --hard
HEAD is now at 69d6f61 Initial commit

C:\Users\...> \Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

```

- a. The *git reset --hard* command has no effect on new untracked files, you can check for yourself. You need *git clean -xfd* to remove them.
- b. Usually when you want to restore the Git repo to only committed changes, you use *git reset --hard* followed by *git clean -xfd*.

11. Now again let's create new file, change existing file, such that both changes are staged, and then we commit them using *git commit -m "First commit message"*:

```

C:\Users\...> \Github\Repos\github-demo-repo>echo "A file" >> a.txt
C:\Users\...> \Github\Repos\github-demo-repo>echo "README.md change 1" >> README.md
C:\Users\...> \Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

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

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

C:\Users\...> \Github\Repos\github-demo-repo>git add -A
C:\Users\...> \Github\Repos\github-demo-repo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md
    new file:   a.txt

C:\Users\...> \Github\Repos\github-demo-repo>git commit -m "First commit message"
[master b326012] First commit message
 2 files changed, 2 insertions(+)
 create mode 100644 a.txt

C:\Users\...> \Github\Repos\github-demo-repo>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

```

- a. You see all changes are committed, but now you're told your local master branch has commits not present in the origin (remote) master branch, and the command *git push* can be used to align them. Syncing local with remote repo will come later, for now we stick to local repo.

12. Check your commits history using *git log --graph --pretty=format:"%h %s"*

A terminal window with a black background and white text. The command `git log --graph --pretty=format:"%h %s"` has been executed. The output shows two commits: `b326012 First commit message` and `69d6f61 Initial commit`. The commit `b326012` is indented relative to `69d6f61`, indicating it is a child commit.

```
* b326012 First commit message
* 69d6f61 Initial commit
```

- a. You can see the new commit `b326012` has a single parent commit `69d6f61` (no line between them using this pretty format). The child-parent relationships between commits will become important soon when you learn about merges.
- b. The alphanumeric commit id presented above is a shorthand, but a much longer full commit id called the SHA1, a 160 bit identifier of the commit usually presented in hexadecimal format.

Now it's time to discuss branches. A *Git branch* is a label of some commit, a friendly name to work with instead of the SHA1 commit id seen above, e.g. "master" is the default branch.

Whenever you commit changes, the current local branch will now point to the new commit. To be more precise, Git maintains a pointer called HEAD which points to current local commit, and when you commit a change both the local branch and HEAD will now point to the new commit. You can also transition to a commit by its SHA1 to modify the HEAD without working on a branch, useful for creating a branch to point that commit you looked up in commits history.

Branches are used to work on multiple features/tasks at the same period, e.g. same 2 weeks sprint or 6 months cycle, such that the changes made to implement each feature/task are isolated from other feature/task until those changes are ready to be merged together.

The recommended branching flow is this:

- *master* branch represents the current stable version of the code in the repo.

- Whenever you want to work on new feature/task/bugfix/etc. you create a new branch pointing to same commit as *master* now points.
- You make changes and commit as much as you need to the new branch.
- You merge the new branch to *master*. It's possible *master* is no longer in same commit as it was when you created the new branch, since someone else may have merged his/her feature/bugfix branch before you merge yours. This might cause a *merge conflict*, shown later, we first start from happy flow as if you're working alone.

13. Create a new branch from current master branch using *git checkout -b <branch name>*, e.g. *git checkout -b feature/task-1*

```
\Github\Repos\github-demo-repo>git checkout -b feature/task-1
Switched to a new branch 'feature/task-1'
```

- This command is shorthand for two commands: *git branch feature/task-1* to create a new branch but keeps you working on current branch, in this example master branch, followed by *git checkout feature/task-1* which transitions your local repo from working on master branch to the new feature branch.

14. Make some changes and commit them, then visualize commits history:

```
\Github\Repos\github-demo-repo>echo "B file" >> B.txt
\Github\Repos\github-demo-repo>git add -A
\Github\Repos\github-demo-repo>git commit -m "Added B.txt for task 1"
[feature/task-1 973fa86] Added B.txt for task 1
1 file changed, 1 insertion(+)
create mode 100644 B.txt
\Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* 973fa86 Added B.txt for task 1
* b326012 First commit message
* 69d6f61 Initial commit
```

- The fact that you transitioned to feature branch doesn't matter for the commits history – you created a new commit that has b326012 as parent, it doesn't matter which branch was used for it, branches are just labels helpful for us, to Git itself they don't matter as far as tracking history, only commits matter.

15. Now transition back to master branch using *git checkout master* and visualize commits history to see the new commit doesn't appear there:

```
C:\Users\johndoe\Documents\Github\Repos\github-demo-repo>git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

C:\Users\johndoe\Documents\Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* b326012 First commit message
* 69d6f61 Initial commit
```

16. Now merge the feature branch to master branch, so the feature branch changes are included in master branch, using *git merge feature/task-1*:

```
C:\Users\johndoe\Documents\Github\Repos\github-demo-repo>git merge feature/task-1
Updating b326012..973fa86
Fast-forward
 B.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 B.txt

C:\Users\johndoe\Documents\Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* 973fa86 Added B.txt for task 1
* b326012 First commit message
* 69d6f61 Initial commit
```

- a. Note there was a “fast forward” merge, this is because the commit pointed to by feature/task-1 branch is direct descendant of the commit pointed to by master branch, so Git realizes the merge the changes to master all that Git has to do is make master branch point to that descendant commit.
 - b. Due to the fast forward behavior, you see the commits history of master branch the same as feature/task-1.
17. Delete branch feature/task-1 since we're done with this task, using *git branch -d feature/task-1*

```
C:\Users\johndoe\Documents\Github\Repos\github-demo-repo>git branch -d feature/task-1
Deleted branch feature/task-1 (was 973fa86).

C:\Users\johndoe\Documents\Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* 973fa86 Added B.txt for task 1
* b326012 First commit message
* 69d6f61 Initial commit
```

- a. Note you could delete the branch because you're not on it, you're on master.
- b. Deleting a branch has no impact on commits history, branches are just labels.

18. Now we simulate changes to master branch while you're working on your feature branches, but changes are for different files so no conflicts. The flow is creating a new feature branch and commit new file, transition back to master and commit new different file, then merge feature branch to master and visualizing commits history:

```
C:\Users\user\Documents>Github\Repos\github-demo-repo>git checkout -b feature/task-2
Switched to a new branch 'feature/task-2'

C:\Users\user\Documents>Github\Repos\github-demo-repo>echo "C file" >> C.txt

C:\Users\user\Documents>Github\Repos\github-demo-repo>git add -A

C:\Users\user\Documents>Github\Repos\github-demo-repo>git commit -m "C file added"
[feature/task-2 2d8f32f] C file added
1 file changed, 1 insertion(+)
create mode 100644 C.txt

C:\Users\user\Documents>Github\Repos\github-demo-repo>git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

C:\Users\user\Documents>Github\Repos\github-demo-repo>echo "D file" >> D.txt

C:\Users\user\Documents>Github\Repos\github-demo-repo>git add -A

C:\Users\user\Documents>Github\Repos\github-demo-repo>git commit -m "D file added"
[master 871bb21] D file added
1 file changed, 1 insertion(+)
create mode 100644 D.txt

C:\Users\user\Documents>Github\Repos\github-demo-repo>git merge feature/task-2
Merge made by the 'recursive' strategy.
C.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 C.txt

C:\Users\user\Documents>Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* da81e9a Merge branch 'feature/task-2'
  * 2d8f32f C file added
  * 871bb21 D file added
/*
* 973fa86 Added B.txt for task 1
* b326012 First commit message
* 69d6f61 Initial commit
```

- a. The merge "recursive" strategy means Git realized master branch and feature/task-2 branches don't point to commits that are direct descendants of one another, but instead they have a shared 3rd ancestor commit, in this case it was commit 973fa86.
- b. You can see the 2d8f32f commit to feature/task-2 branch, and the separate 871bb21 commit to master branch, and then the new *merge commit* da81e9a.

- c. A *merge commit* is a special kind of commit that is the result of a merge that wasn't fast forwarded, and therefore has two parents, not one.

19. Now we simulate a merge conflict by doing the same thing only where in master and the feature branch we change the same file instead of adding new files (we could also add new files with same name):

```
Github\Repos\github-demo-repo>git branch -d feature/task-2
Deleted branch feature/task-2 (was 2d8f32f).

Github\Repos\github-demo-repo>git checkout -b feature/task-3
Switched to a new branch 'feature/task-3'

Github\Repos\github-demo-repo>echo "A new file" >> A.txt

Github\Repos\github-demo-repo>git add -A

Github\Repos\github-demo-repo>git commit -m "Changing A.txt file for task 3"
[feature/task-3 874c5b9] Changing A.txt file for task 3
1 file changed, 1 insertion(+)

Github\Repos\github-demo-repo>git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 5 commits.
(use "git push" to publish your local commits)

Github\Repos\github-demo-repo>echo "Another new file" >> A.txt

Github\Repos\github-demo-repo>git add -A

Github\Repos\github-demo-repo>git commit -m "Changing A.txt file for master"
[master 3ae5685] Changing A.txt file for master
1 file changed, 1 insertion(+)

Github\Repos\github-demo-repo>git merge feature/task-3
Auto-merging a.txt
CONFLICT (content): Merge conflict in a.txt
Automatic merge failed; fix conflicts and then commit the result.
```

20. We got a merge conflict, we need to fix it. There are several tools that help to merge conflicts, with useful visualization options to see conflicted changes, I use VS but there's also KDiff3 and you can google for other Git merge tools and how to configure them as your default merge tool. For demo purposes choose to take the file content as it is in master branch using *git checkout --ours*.

```

Github\Repos\github-demo-repo>git status
On branch master
Your branch is ahead of 'origin/master' by 6 commits.
(use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   a.txt

no changes added to commit (use "git add" and/or "git commit -a")
Github\Repos\github-demo-repo>git checkout --ours .
Updated 1 path from the index
Github\Repos\github-demo-repo>git add -A
Github\Repos\github-demo-repo>git status
On branch master
Your branch is ahead of 'origin/master' by 6 commits.
(use "git push" to publish your local commits)

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

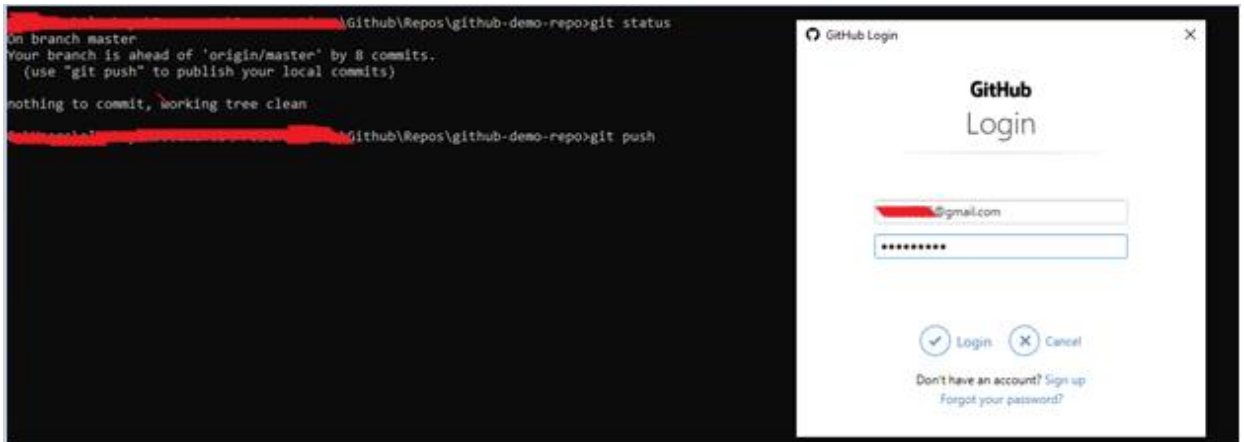
Github\Repos\github-demo-repo>git commit -m "Merged A.txt file from feature/task-3 to master"
[master f8e2407] Merged A.txt file from feature/task-3 to master
Github\Repos\github-demo-repo>git log --graph --pretty=format:"%h %s"
* f8e2407 Merged A.txt file from feature/task-3 to master
* 874c5b9 Changing A.txt file for task 3
* 3ae5685 Changing A.txt file for master
* da81e9a Merge branch 'feature/task-2'
* 2d8f32f C file added
* 871bb21 D file added
* 973fa86 Added B.txt for task 1
* b326012 First commit message
* 69d6f61 Initial commit

```

- a. You can see the status outputs during different steps of handling the conflict.
- b. You can see the merge commit, the fact there were conflicts doesn't change how the commit history will look like.
- c. Always resolve conflicts using a proper merge tool in real scenario! If your merge is bad, you'll need to revert it, see the last section of this walkthrough.

Now's the time to work with remote repo at last. Three major commands exist, one to align remote repo with local changes called *git push*, and the others to align local repo with remote changes called *git fetch* and *git pull*. Both those commands work by comparing local branch to matching remote branch.

21. Push master branch using *git push*, requires working internet connection, you may be prompted for login the first time:



```
Github\Repos\github-demo-repo>git push
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (22/22), 1.86 KiB | 634.00 KiB/s, done.
Total 22 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), done.
To https://github.com/[redacted]/github-demo-repo.git
69d6f61..f8e2407 master -> master
```

- a. Once commits are pushed you NEVER modify them locally, they become immutable at that point! There are advanced Git techniques mentioned below to modify commits, be very careful with them to avoid what's known as *Git hell*: If you modify pushed commits then the commits history in local repo and remote repo are no longer aligned, and that's very bad, at that point it's like your smartphone no longer starts – either you take it to a lab or you replace it – with Git repos unless you're a real expert and know what you're doing you better off deleting your local repo and re-cloning at that point.

22. The 2nd time you push you shouldn't be prompted for login, if you do then look up "git credentials manager" and download for [Windows](#)/[Mac](#)/other:

```
\Github\Repos\github-demo-repo>echo "A really new file" >> A.txt
\nGithub\Repos\github-demo-repo>git add -A
\nGithub\Repos\github-demo-repo>git commit -m "Modified A.txt again to push 2nd time"
[master a76a39b] Modified A.txt again to push 2nd time
1 file changed, 1 insertion(+)
\nGithub\Repos\github-demo-repo>git push
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), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/.../github-demo-repo.git
 f8e2407..a76a39b master -> master
```

23.Look in Github and see the committed files by navigating to your repo:

master ▾ 1 branch 0 tags Go to file Add file ▾ Code ▾

Eldar Sehayek Modified A.txt again to push 2nd time a76a39b 1 minute ago 10 commits

B.txt	Added B.txt for task 1	1 hour ago
C.txt	C file added	42 minutes ago
D.txt	D file added	42 minutes ago
LICENSE	Initial commit	2 hours ago
README.md	First commit message	1 hour ago
a.txt	Modified A.txt again to push 2nd time	1 minute ago

README.md

github-demo-repo

Preparation exercise for Github demo "README.md change 1"

- Github doesn't offer commits history visualization like some other publicly available origin servers, e.g. Azure DevOps (abbr. ADO), you need to use local visualization tools for that, popular ones mentioned below.

24.Now create a new branch in Github itself, add new file and commit:

🔗 master ▾ 🔗 1 branch 🏷️ 0 tags

Switch branches/tags

feature/task-4

Branches Tags

🔗 Create branch: feature/task-4 from 'master'

[View all branches](#)

📄 LICENSE

🔗 feature/task-4 ▾ 🔗 2 branches 🏷️ 0 tags

[Go to file](#) [Add file ▾](#) [📄 Code ▾](#)

This branch is even with master. 🔗 Pull request ⚖️ Compare

[Go to file](#) [Add file ▾](#) [📄 Code ▾](#)

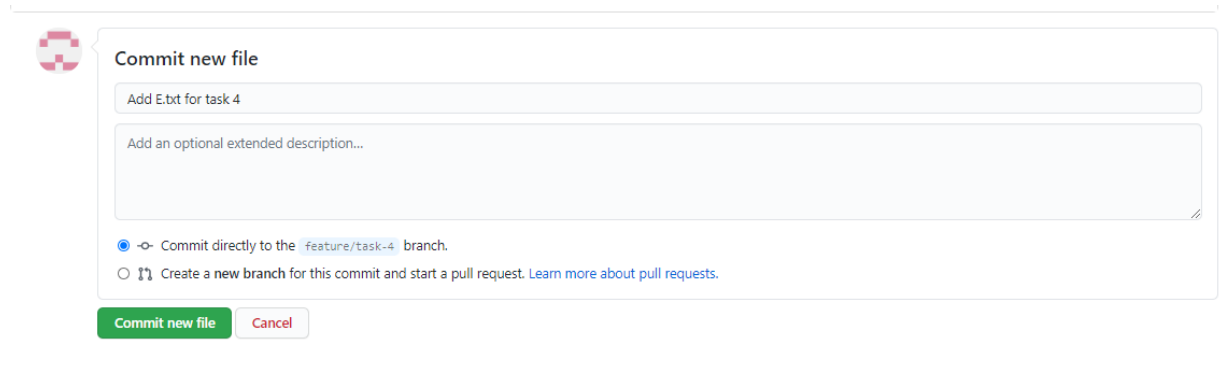
[Create new file](#)
[Upload files](#)

⚖️ Compare

github-demo-repo / [Cancel](#)

<> Edit new file 👁️ Preview

```
1 E.txt file for task 4
```

25. Now back to local repo, we fetch remote repo commits & branches using *git fetch* then we checkout the new branch:

```
C:\Users\user> cd C:\Github\Repos\github-demo-repo && git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 654 bytes | 54.00 KiB/s, done.
From https://github.com:github-demo-repo
* [new branch]      feature/task-4 -> origin/feature/task-4

C:\Users\user> cd C:\Github\Repos\github-demo-repo && git checkout feature/task-4
Switched to a new branch 'feature/task-4'
Branch 'feature/task-4' set up to track remote branch 'feature/task-4' from 'origin'.
```

- a. The *git fetch* command syncs local repo with new commits & branches from remote repo but doesn't change your local branch nor HEAD pointer. It's needed in this case so we can transition to a branch that didn't yet exist locally.

26. Now commit to the remote branch again to same file, then locally use *git pull* to sync remote branch changes to local branch changes, and visualize commits history:


```

C:\Users\user>cd C:\Users\user\Github\Repos\github-demo-repo>git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 670 bytes | 55.00 KiB/s, done.
From https://github.com/user/github-demo-repo
   83760a7..f31c2e0  feature/task-4 -> origin/feature/task-4
Updating 83760a7..f31c2e0
Fast-forward
 E.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\Users\user>cd C:\Users\user\Github\Repos\github-demo-repo>git log --graph --pretty-format:"%h %s"
* f31c2e0 Modified E.txt file for task 4 to pull
* 83760a7 Add E.txt for task 4
* a76a39b Modified A.txt again to push 2nd time
* f8e2407 Merged A.txt file from feature/task-3 to master
//
* 874c5b9 Changing A.txt file for task 3
* 3ae5685 Changing A.txt file for master
//
* da81e9a Merge branch 'feature/task-2'
//
* 2d8f32f C file added
* 871bb21 D file added
//
* 973fa86 Added B.txt for task 1
* b326012 First commit message
* 69d6f61 Initial commit

```

- a. You see “fast forward” in the pull output because in fact remote branches are represented as local branches with naming convention “origin/<branch-name>” so they aren’t confused with local branch. The pull fetches commits to this local branch representing remote branch, then merges it to actual local branch, and there might be merge conflicts to resolve.
- b. Never modify local branches representing origin directly! Don’t commit to them, let Git modify them when you fetch/pull or as part of other commands.

That’s it! Now you know basics to work alone with Git & Github, committing changes, using branches, merging them, sync local repo with remote, you’re set!

Next steps

1. To work with others, you should learn about Github:
 - Pull requests (abbr. PRs) and how to perform code reviews (abbr. CRs) for pull requests.
 - Branch policies to prevent push to master, and control merge types allowed to master.
 - More worthwhile Github settings and features.
2. There’s more to explore with Git for more advanced usages, for example:
 - *git revert* - Reverting a commit that caused a regression; be careful when reverting a merge commit, it has two parent commits so make sure to revert to the correct one!

- *git stash* - Stashing changes that you don't want to commit but also don't want to lose. Can be done with temp local branch but there's a stashing feature to not pollute local repo with branches you don't need.
- *git tag* - Tagging stable code versions
- [Squashing commits](#) locally before pushing so you don't push too many commits that are uninteresting in the remote repo; be careful to never squash commits already pushed, don't modify commits in any way if they are in origin!

3. There are also very useful tools to work with Git, cmd/bash is basics:

- Every popular IDE has Git & Github support, may require plugin.
- Github Desktop for working on local repos with Github origin.
- SourceTree from Atlassian, provides UI for local Git repos.
- TortoiseGit lets you work with Git in the file explorer itself
- Powershell has posh-git, very useful git module that visualizes current branch, tracked/untracked files, etc.
- Etc.