



# git introduction

Seda Radojkova  
Daniel Molinuevo

September 2025

# What is git?

Git is a distributed version control software system that is capable of managing versions of source code or data. It is often used to control source code by programmers who are developing software collaboratively.

Two most common git softwares

Used in BIO-210



We want to avoid this:

---

📄 project\_copy\_copy.py

---

📄 project\_copy.py

---

📄 project\_dani\_v1\_seda\_change.py

---

📄 project\_dani\_v1.py

---

📄 project\_final\_final\_for\_real.py

---

📄 project\_final\_final.py

---

📄 project\_final\_seda\_final\_final.py

---

📄 project\_final.py

---

📄 project\_seda\_copy.py

---

📄 project\_seda.py

---

📄 project.py

---



DON'T DO THIS

# Why git?

- Version control
  - Keeps track of **all past versions** of the repo/files
  - Take home: any change done via git commands can always be **recovered!**
  
- Collaborative work
  - Easy to work **in parallel**
  - Easy to **review** and **integrate** other's code
  
- Open source :)

# What we are doing today

- Live demo!
- We will go through some of the basic git commands:
  - Basic commands
  - Branches
  - Comitting and pushing changes
  - Git pull
  - Merging branches
  - Others (reset, stash, restore)

We avoid this!



Do not blindly copy paste commands, ask us!



## Basic commands Setting up

`git status`

`git clone`

`git init`



Provides general info on the current repository (e.g files modified, current branch, etc). ALWAYS, carefully **read** its output

```
git status
fatal: not a git repository (or any of the parent directories): .git
```

this terminal is not inside a git repository

```
~/Desktop/github-demo > git main git status
On branch main          ← "main" = name of branch
No commits yet          ← status of the branch
nothing to commit (create/copy files and use "git add" to track)
```

any changes that have been made on the branch

```
Apple > ~/Desktop/github-demo > git main ?3 git status
On branch main ← where are we (which branch)

No commits yet ← what have we done (nothing in this case)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  README.md
  main.py
  src/
} ← files that are not being "tracked" (i.e. version controlled);
  they have been created/edited but have not been
  "added" and "committed" (more on this in a bit)

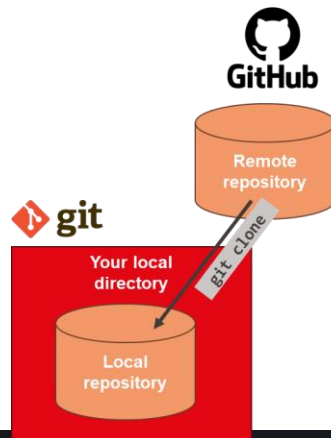
nothing added to commit but untracked files present (use "git add" to track)
← status of "added" files that are about to be "committed"
```

In case of doubt: `git status`

# git clone

Used to "download" (clone) a **remote** repository to the **current folder**.

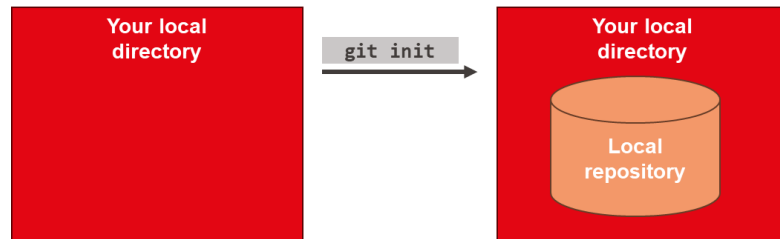
```
Apple > ~/Desktop git clone https://github.com/dmg99/github-demo
Cloning into 'github-demo'...
warning: You appear to have cloned an empty repository.
```



If it is a **private** repo, you need to have been granted **access**, otherwise you will get a "not found" error:

```
Cloning into 'github-demo2'...
remote: Repository not found.
fatal: repository 'https://github.com/dmg99/github-demo2/' not found
```

# git init + git remote



Alternative to “`git clone`”. Instead of downloading a repository, it **initializes** one in the **current folder** (it is a bit more messy).

```
➤ ~ > ~/Desktop/github-demo2 git init
Initialized empty Git repository in /Users/molinuev/Desktop/github-demo2/.git/
```

# git init + git remote

Alternative to “git clone”. Instead of downloading a repository, it **initializes** one in the **current folder** (it is a bit more messy).

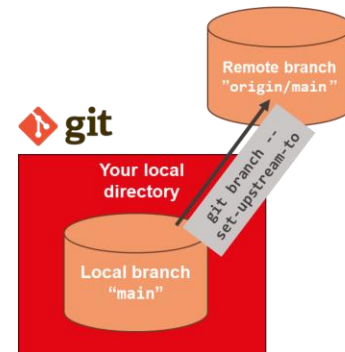


```
➤ 🍏 > ➤ ~/Desktop/github-demo2 git init
Initialized empty Git repository in /Users/molinuev/Desktop/github-demo2/.git/
```

Need to link local repository to remote repository using “git remote add”

```
➤ 🍏 > ➤ ~/Desktop/github-demo2 > git main git remote add origin https://github.com/dmg99/github-demo
```

# git init + git remote



Alternative to “git clone”. Instead of downloading a repository, it **initializes** one in the **current folder** (it is a bit more messy).

```
🍏 > ~/Desktop/github-demo2 git init
Initialized empty Git repository in /Users/molinuev/Desktop/github-demo2/.git/
```

Need to link local repository to remote repository using “git remote add”

```
🍏 > ~/Desktop/github-demo2 > git main git remote add origin https://github.com/dmg99/github-demo
```

Need to link the **current branch** to a **remote one**

```
git branch --set-upstream-to=origin/main main
```

```
> git checkout -b new_branch  
> git merge new_branch
```



## Branches

Local vs remote

```
git branch
```

```
git checkout
```

When speaking of branches, we need to distinguish:

- **Remote branches:** they exist in the “remote” repository (e.g on GitHub)
- **Local branches:** they exist in the **current folder** we are working on (e.g our PC).

A local branch **can be set to track** a remote branch. This means that it will **pull and push** to that one. *Note that this means that **local branches can exist without a remote counterpart.***

If we try to push/pull in a local branch without remote:

```
└─ Apple > ~/De/github-demo > git my_local_branch !1 git pull
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details. ← get more help

git pull <remote> <branch> ← we need to know from which:
                             (1) repository and (2) branch

If you wish to set tracking information for this branch you can do so with:

git branch --set-upstream-to=origin/<branch> my_local_branch ← we just did this in
                                                                the previous slide!
```

Lists existing **local** branches

```
~/De/github-demo > git branch
```

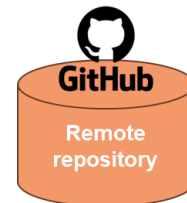
```
* featureA
  main
(END)
```



Flag `--remote` allows to list also **remote** branches

```
git branch --remote
```

```
origin/featureA
origin/featureB
origin/main
(END)
```



Changes current working branch.

If branch does not exist locally but there is a remote with that name (*need to run `git fetch` first*) it will **create a new one and track it**.

```
▶ Apple > ~/Desktop/github-demo > main !1 git checkout featureA
branch 'featureA' set up to track 'origin/featureA'. ← create new local
Switched to a new branch 'featureA' ← switch to the newly created branch
                                                    branch; note: it
                                                    tracks "origin/main"
```

If it exists locally, it will simply **switch branch**

```
▶ Apple > ~/De/github-demo > featureA !1 git checkout main
Switched to branch 'main' ← switch to the local branch
Your branch is up to date with 'origin/main'. ← branch status
```

**Note: to checkout branch, you need to commit/stash your current changes**

# git checkout – create branch

When using the `-b` flag, checkout will create a **new local branch** that **forks from the current one**

```
Apple > ~/Desktop/github-demo > git checkout -b new_branch  
Switched to a new branch 'new_branch'
```

If a **remote one does not exist**, when **pushing for the first time** you need to run the following:

```
Apple > ~/De/github-demo > git branch -u origin/featureA new_branch  
branch 'new_branch' set up to track 'origin/featureA'.
```

This would **pair the local branch “new\_branch”** with the **remote branch “featureA”**.

git commit



git push



git add .



## Committing changes

git add

git rm

git commit

git push

Used to prepare a “commit”.

By using “git add” we **specify which files** we would like to commit

```
➤ ~ / Desktop / github-demo > git main ?3 git add README.md
➤ ~ / Desktop / github-demo > git main +1 ?2 git status
```

On branch main ← where are we (which branch)

No commits yet ← what have we done (nothing in this case)

Changes to be committed:  
(use "git rm --cached <file>..." to unstage) ← think: git adds files "to the stage"

new file: README.md ← what are we about to commit

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

main.py ← these files are being ignored for now  
src/

To add **all modified files**, you can use “git add .”

Be **CAREFUL** when doing this and **always check** with “git status” which files have been included.

```
└─ Apple > ~/Desktop/github-demo > git main +1 ?2 git add .
└─ Apple > ~/Desktop/github-demo > git main +4 git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md
    new file:   main.py
    new file:   src/extract_data.py
    new file:   src/process_files.py
```

DOT  
adds *all*  
modified  
files



# git commit

Committing is one of the main actions in git.

It consists of **taking a set of changes already specified** via “git add” and “git rm” and **putting them together**.

```

└─ Apple > ~ / Desktop / github-demo > main +1 git commit -m "Added description to README.md"
[main 848a973] Added description to README.md
 1 file changed, 3 insertions(+)
└─ Apple > ~ / Desktop / github-demo > main :1 git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean
  
```

Annotations:

- `-m "Added description to README.md"`: `-m` for message and the message is in ""
- “Your branch is ahead of 'origin/main' by 1 commit.”: what have we done *locally*
- “nothing to commit, working tree clean”: any uncommitted changes?

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAHAHAHAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

PSA: write **MEANINGFUL**  
commit messages  
**CONSISTENTLY**

Cartoon credits: <https://xkcd.com/1296/>

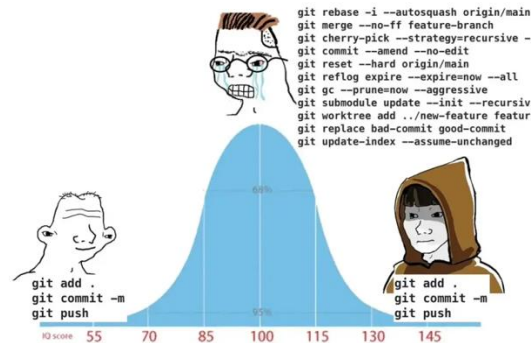
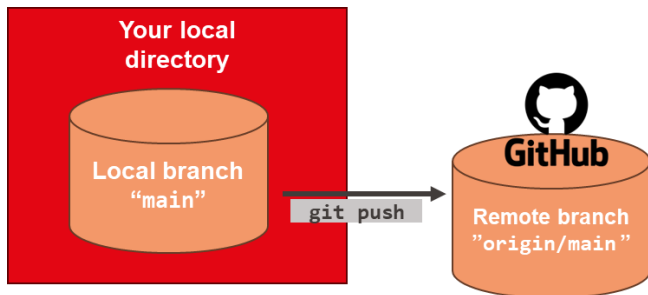
# git push

Used to **sync the local changes** (commits) with the **remote**.

Note that it will **only push committed changes!**

```

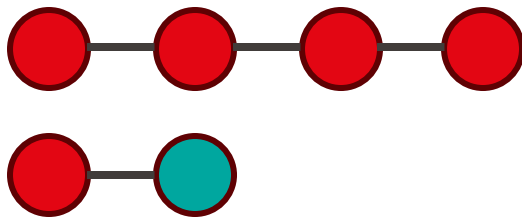
> ~/Desktop/github-demo > main #1 git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 405 bytes | 405.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/dmg99/github-demo
544708e..848a973 main -> main
  
```



In order to push, the *current branch has to be synced with the remote!*

```

🍏 > ~ / Desktop / github-demo2 > 🐛 main ↕2↕1 git push
To https://github.com/dmg99/github-demo
! [rejected]      main -> main (non-fast-forward) ← divergence error
error: failed to push some refs to 'https://github.com/dmg99/github-demo'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again. ← sync branches
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
  
```



remote branch

local branch

missing commits from remote  
and has a different commit



Once you **push to remote**, all your changes **will be saved!**



Git pull



## Syncing remote changes to local

```
git pull
```

```
git pull --rebase
```

```
git fetch
```

Used to **update** the changes (commits) **from the remote branch to the local branch**. After running it, mainly 4 things can happen:

1. **Up to date:** No new changes in the remote branch

```
Apple > ~/Desktop/github-demo > main !1 git pull  
Already up to date.
```

2. **Successful pull:** There are changes and local branch can be easily updated

```
Apple > ~/Desktop/github-demo > main ↓1 !1 git pull  
Updating 848a973..9740ded  
Fast-forward  
main.py | 2 ++  
1 file changed, 2 insertions(+)
```

Used to **update** the changes (commits) **from the remote branch to the local branch**. After running it, mainly 4 things can happen:

**3. Merge conflict:** both you and the remote **have modified the same file**.

Two options to fix this: (1) `git stash` or (2) `git commit + git pull --rebase`

```

$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 356 bytes | 118.00 KiB/s, done.
From https://github.com/dmg99/github-demo
   9740ded..6241bec  main    -> origin/main
Updating 9740ded..6241bec
error: Your local changes to the following files would be overwritten by merge:
   main.py
Please commit your changes or stash them before you merge.
Aborting

```

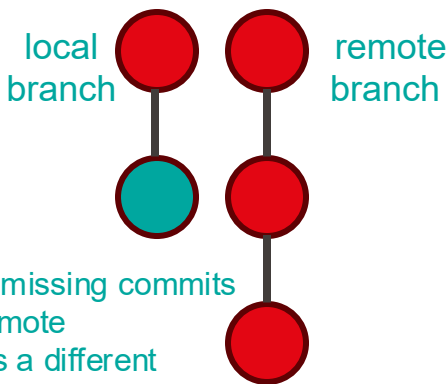
remote branch

local branch with uncommitted changes

`git commit + git pull --rebase`      `stash = "put on hold until you sort out pulling"`

4. **Diverged branches:** you have **local commits not present in remote** and **remote has commits not present in local**.

**Fix:** Try `git pull --rebase`. If it does not work, probably both **the local and remote changes modified the same files**. Then you need to **merge conflicts** (big mess up, ask TA/SA)



```

🍏 > ~/Desktop/github-demo > 📄 main :1:1 git status
On branch main
Your branch and 'origin/main' have diverged, ← divergence warning
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
nothing to commit, working tree clean ← note that git pull merges changes from the
remote branch into the local one by default
🍏 > ~/Desktop/github-demo > 📄 main :1:1 git pull

hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false # merge
hint:   git config pull.rebase true  # rebase
hint:   git config pull.ff only     # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
  
```

help message: configure default behaviour

pass --rebase to git pull aka suggests running git pull --rebase

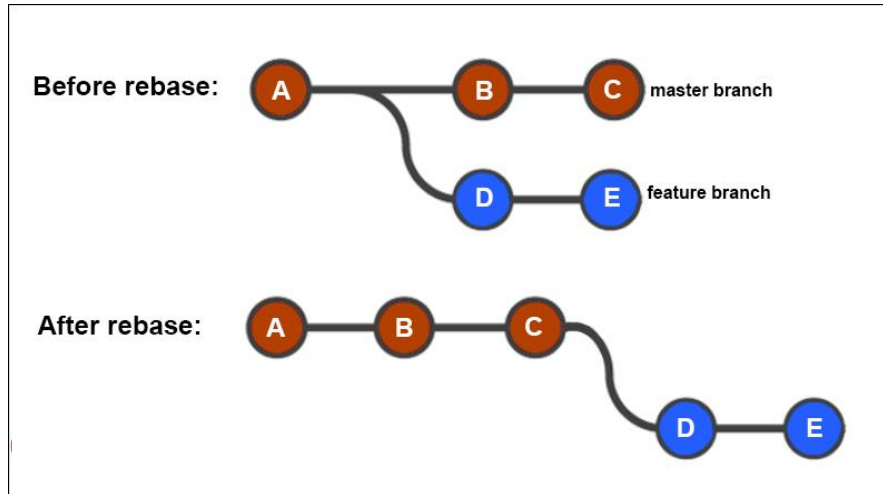
In case the local branch and remote branch **have diverged** but their commits **modify different files** (or **different parts** of the **same file**), **rebasing** will **stack** the local **commits** on top of the remote ones.

```
🍏 > ~/Desktop/github-demo > 🐱 main ↓1↑1 git status
On branch main
Your branch and 'origin/main' have diverged, ← divergence warning
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
🍏 > ~/Desktop/github-demo > 🐱 main ↓1↑1 git pull --rebase
Successfully rebased and updated refs/heads/main.
🍏 > ~/Desktop/github-demo > 🐱 main ↑1 git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit. ← rebase worked and
(use "git push" to publish your local commits) needs to be pushed to be
visible on the remote repo

nothing to commit, working tree clean
```

Rebasing can be represented as the following operations on the chain of commits.



We would say that:  
Branch “feature” has  
been **rebased to**  
branch “master”

Allows to **check for changes** in remote. It looks for **new branches** and **whether there are changes** in the existing ones.

Unlike `git pull`, it ***does not update local files***.

```
➤ 🍏 > ~/De/github-demo2 > 🐙 📄 main :2:1 +1 git fetch
From https://github.com/dmg99/github-demo
* [new branch]      featureA    -> origin/featureA
* [new branch]      featureB    -> origin/featureB
```

**Note:** git will have **no clue** of what is happening **in remote** if you do not run `git fetch`.

If for example you run `git status` and you get “up to date with remote” this will only make sense if you have run `git fetch` first. Otherwise there might be changes that git is not “seeing”.



# Merging branches Pull requests

Merge – Pull request  
git rebase

# Merging branches – Pull requests

When you are **done working on a branch** (e.g featureA branch) and you want to **merge it to main**, the best way is to **create a pull request** on GitHub.

The screenshot displays the GitHub interface for a repository. At the top, the repository name 'dmg99 / github-demo' is visible. Below the navigation bar, the 'Pull requests' tab is selected. A search bar contains the filter 'is:pr is:open'. To the right, there are buttons for 'Labels 9' and 'Milestones 0'. A prominent green button labeled 'New pull request' is highlighted with a red rectangular box. Below this, a large white box contains the text: 'Welcome to pull requests! Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#).' At the bottom of the page, a 'ProTip!' suggests mixing filters to narrow down results. The footer contains the GitHub logo, copyright notice '© 2025 GitHub, Inc.', and links for Terms, Privacy, Security, Status, Community, Docs, Contact, Manage cookies, and Do not share my personal information.

dmg99 / github-demo

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base: main ← compare: featureB

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) [Create pull request](#)

3 commits 2 files changed 1 contributor

Commits on Sep 25, 2025

- made function to process files** 79adce2 <>  
dmg99 committed 3 hours ago
- updated README** 1f87b1f <>  
dmg99 committed 2 hours ago
- updated process file function** 5d2731c <>  
dmg99 committed 2 hours ago

Showing 2 changed files with 3 additions and 1 deletion. Split Unified

2 README.md

```
... .. @@ -1,3 +1,3 @@
1 1 # Github demo repo
2 2
3 - This repo will be used as an example for the BI0-210 course
3 + This repo will be used as an example for the BI0-210 course,
```

# Merging branches – Pull requests

The screenshot shows the top navigation bar of a GitHub repository. On the left, there is a hamburger menu icon, a profile picture, and the repository name 'dmg99 / github-demo'. On the right, there is a search bar with the placeholder text 'Type / to search' and several utility icons: a repository icon, a refresh icon, a plus icon, a refresh icon, a search icon, and a mail icon. Below the navigation bar, there is a secondary navigation bar with links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Pull requests' link is highlighted with a red underline.

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

The screenshot shows the 'Open a pull request' form in GitHub. The form is divided into several sections:

- Base and Compare:** A section with two dropdown menus. The first is labeled 'base: main' and the second is labeled 'compare: featureB'. This section is highlighted with a red box.
- Add a title:** A text input field containing the text 'Finished implementing the data processing'. This field is highlighted with a red box.
- Add a description:** A rich text editor with a 'Write' tab selected. The text area contains the placeholder text 'Add your description here...'. Below the text area, there are two checkboxes: 'Markdown is supported' and 'Paste, drop, or click to add files'. This section is highlighted with a red box.
- Reviewers and Assignees:** A sidebar on the right side of the form. It includes sections for 'Reviewers', 'Suggestions', 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Development'. The 'Reviewers' section is currently empty. The 'Assignees' section shows 'No one—assign yourself'. The 'Labels' section shows 'None yet'. The 'Projects' section shows 'None yet'. The 'Milestone' section shows 'No milestone'. The 'Development' section shows 'Use [Closing keywords](#) in the description to automatically close issues'.
- Helpful resources:** A section at the bottom right with the text 'Helpful resources' and a link to 'GitHub Community Guidelines'.

At the bottom right of the form, there is a green button labeled 'Create pull request' with a dropdown arrow. This button is highlighted with a red box.

At the bottom left of the form, there is a small icon and the text 'Remember, contributions to this repository should follow our [GitHub Community Guidelines](#)'.



dmg99 / github-demo

Search: Type to search

Code Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

## Finished implementing the data processing #1

Edit Code

Open dmg99 wants to merge 3 commits into main from featureB

Conversation 0 Commits 3 Checks 0 Files changed 2

+3 -1



dmg99 commented now

No description provided.



dmg99 added 3 commits 3 hours ago

made function to process files

79adce2

updated README

1f87b1f

updated process file function

5d2731c



**No conflicts with base branch**  
Merging can be performed automatically.

Merge pull request You can also merge this with the command line. [View command line instructions.](#)

Reviewers  
Suggestions  
Copilot Request

Still in progress? [Convert to draft](#)

Assignees  
No one—assign yourself

Labels  
None yet

Projects  
None yet

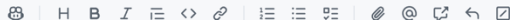
Milestone  
No milestone

Development  
Successfully merging this pull request may close these issues.  
None yet



Add a comment

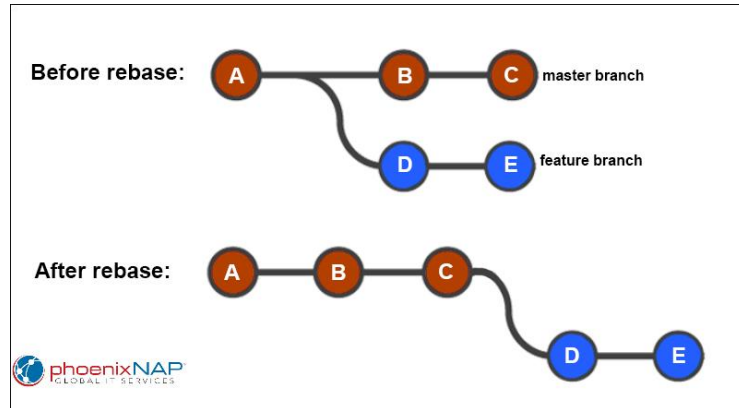
Write Preview



If you have two **local** branches and you want to rebase one onto another, you can use “git rebase”.

For example, if I am working on featureB branch and I want to update it with some changes made to main, I can run:

```
Apple > ~/Desktop/github-demo > featureB git rebase main featureB  
Successfully rebased and updated refs/heads/featureB.
```



This would “rebase featureB onto main”.

In general, you should **never** have to use this command.

The only case where you should find the need for this is if you have **rebased a local branch** (e.g. featureB) onto another (e.g. main) and you want to **push this change to the remote featureB**.

If you try to push normally, you might get:

```
🍏 > ~/De/github-demo > featureB .1.2 git push
To https://github.com/dmg99/github-demo
 ! [rejected]        featureB -> featureB (non-fast-forward) ← divergence error
error: failed to push some refs to 'https://github.com/dmg99/github-demo'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again. ← by default, remote takes precedence
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

What the `--force` flag does is **overwriting previous history of the remote chain** (potentially very dangerous!).

If you have **previously rebased** it makes sense to do so, because when rebasing you **have overwritten past commits** in the chain.

```
Apple > ~/De/github-demo > P featureB .1:2 git push --force
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 686 bytes | 686.00 KiB/s, done.
Total 7 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/dmg99/github-demo
+ 5198987...5d2731c featureB -> featureB (forced update)
```

**Note: ONLY USE --force IN THIS USE CASE.**

If you are in another situation and think that `--force` is the solution to your problem, you are probably wrong, ask a TA/SA!

# Other useful commands

```
git reset
```

```
git restore
```

```
.gitignore
```

git reset can be used to **go backwards** in the commit chain (**locally!**).

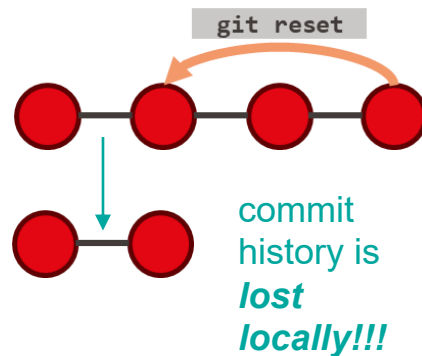
**Example 1:** remove local changes made to **tracked** files.

```
🍏 > ~/Desktop/github-demo > 🐙 featureB !1 git status
On branch featureB
Your branch is up to date with 'origin/featureB'.

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

no changes added to commit (use "git add" and/or "git commit -a")
🍏 > ~/Desktop/github-demo > 🐙 featureB !1 git reset --hard
HEAD is now at 5d2731c updated process file function
🍏 > ~/Desktop/github-demo > 🐙 featureB git status
On branch featureB
Your branch is up to date with 'origin/featureB'.

nothing to commit, working tree clean
```



**--hard means all commit history is lost locally!!!**

**Example 2:** remove local commit(s) and **reset head to the origin commit.**

```

🍏 > ~/Desktop/github-demo > 🐱 main ↑4 git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

🍏 > ~/Desktop/github-demo > 🐱 main ↑4 git reset --hard origin/main
HEAD is now at 12a642b made function to extract data

🍏 > ~/Desktop/github-demo > 🐱 main git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

```

Diagram illustrating the effect of `git reset --hard origin/main`. The diagram shows a **remote branch** (top) with two red circles representing commits, and a **local branch** (bottom) with four red circles. An orange arrow labeled `git reset` points from the local branch back to the first commit of the remote branch. A blue arrow points to the text `--hard means local commit history is lost!!!`.

**Note:** be careful with this command. It is surprisingly easy to “get lost” in the chain. **Moreover the changes are lost for good!**

The main flags used with `git reset` are

- **Soft** (`--soft`): if you have staged changes, it will **keep** them.
- **Hard** (`--hard`): if you have staged changes, it will **discard** them.
- **Mixed** (`--mixed`): if you have staged changes, it will **unstage** them but **keep the files**.

Choosing “where” to go can be done as follows:

- **Commit hash**: will go to a specific commit (e.g. `git reset 12a642b`)
- **Head of a branch**: e.g. `git reset origin/main` will reset to the head of the remote main
- **Backwards by n commits**: you can use “`git reset HEAD~n`”, e.g. to go back two commits you can do “`git reset HEAD~2`”

# git log --oneline

Useful to check the **commit history**, good combo with reset.

```
12a642b (HEAD -> main, origin/main, origin/HEAD) made function to extract data
6241bec added main.py function!
9740ded added imports in main.py
848a973 (origin/featureA) Added description to README.md
544708e Initializing repo with basic files
(END)
```

↑  
commit  
hashes

↑  
commit  
messages

`git restore` can be used with the staged flagged in case you have **added a file to commit by mistake**. Note: this can be used **before** committing.

```
Apple > ~/Desktop/github-demo > featureB +1 git status
On branch featureB
Your branch is up to date with 'origin/featureB'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   src/extra_file.py ← accidentally added this file to the stage

Apple > ~/Desktop/github-demo > featureB +1 git restore --staged src/extra_file.py
Apple > ~/Desktop/github-demo > featureB ?1 git status
On branch featureB
Your branch is up to date with 'origin/featureB'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    src/extra_file.py ← file is no longer staged but remains in local directory

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

This is used "save apart" **some unstaged/uncommitted changes** so that they can **later be retrieved and committed**. Its use is:

```
🍏 > ~/De/github-demo > 🐱 main !1 git status
On branch main
Your branch is up to date with 'origin/main'.

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  ← file is modified but not committed

no changes added to commit (use "git add" and/or "git commit -a")
🍏 > ~/Desktop/github-demo > 🐱 main !1 git stash
Saved working directory and index state WIP on main: 12a642b made function to extract data
🍏 > ~/Desktop/github-demo > 🐱 main *1 git status
On branch main
Your branch is up to date with 'origin/main'. ← post stash, no changes are visible

nothing to commit, working tree clean
```

Stashes are **similar to commits**, since they contain **changes** and have a specific **ID**. You can list the local stashes by using “`git stash list`”:

```
stash@{0}: WIP on main: 12a642b made function to extract data  
(END)
```

Used to **recover the last stash**. Note that it *could lead to merge conflicts* if the stash modifies a file already modified in your local branch.

```
Apple > ~/De/github-demo > main *1 git stash pop
On branch main
Your branch is up to date with 'origin/main'.

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 ← changes are retrieved but still need to
                                be staged ("added") and committed
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (3f5394a4f7320ef55f966806a9817c48fc2fad2b)
```

**Note:** this will **bring back the changes** and then **delete** the stash!

**Note2:** if you have multiple you can choose to “pop” a specific stash with “git stash pop stash@{n}” where “n” has to be replaced.

# .gitignore file

This is a file that you can create in your repo so that git **completely ignores some files** in the repo. It looks something like this:

```
1  # Ignore data outputs
2  *.csv
3  *.tsv ← ignore a pattern
4
5  # Ignore log files
6  *.log
7
8  # Ignore large datasets folder
9  /data/ ← ignore a directory and
10         its contents
11 # Ignore OS or editor files
12 .DS_Store ← can add
13 *.swp      comments
14            using #
```

For the specific syntax, go to:

<https://git-scm.com/docs/gitignore>

This file is usually **created at the beginning of a project** and then updated when necessary.