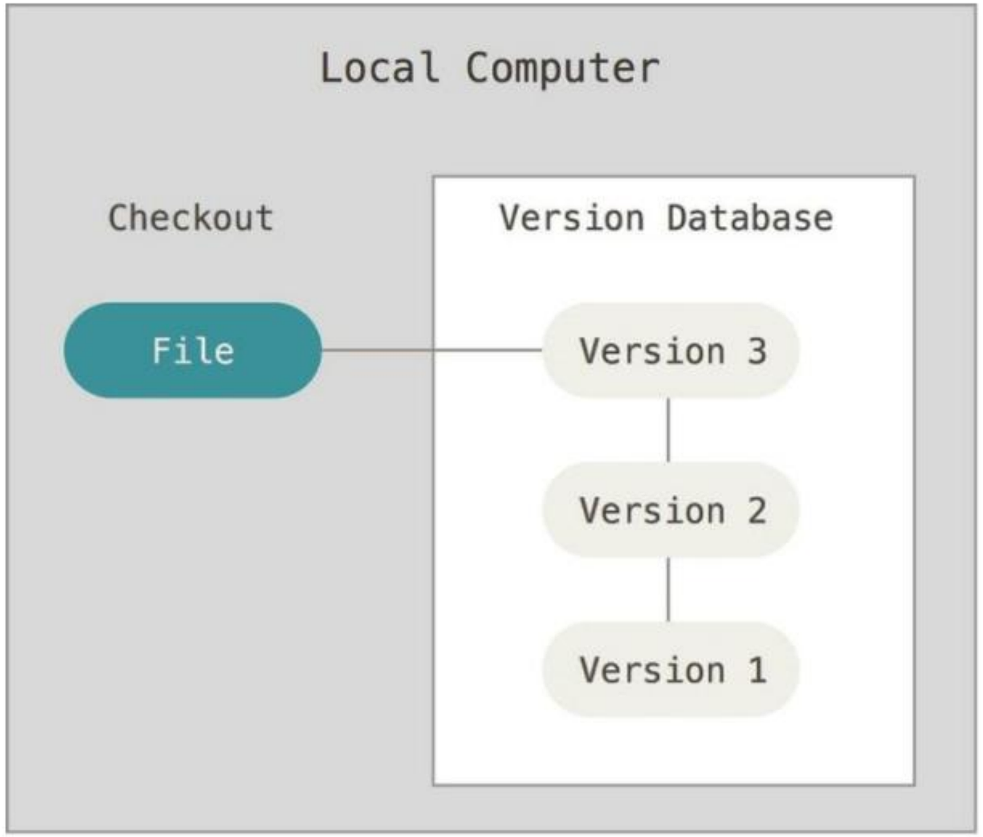
PRACTICE ENTERPRISE

Git

VERSION CONTROL

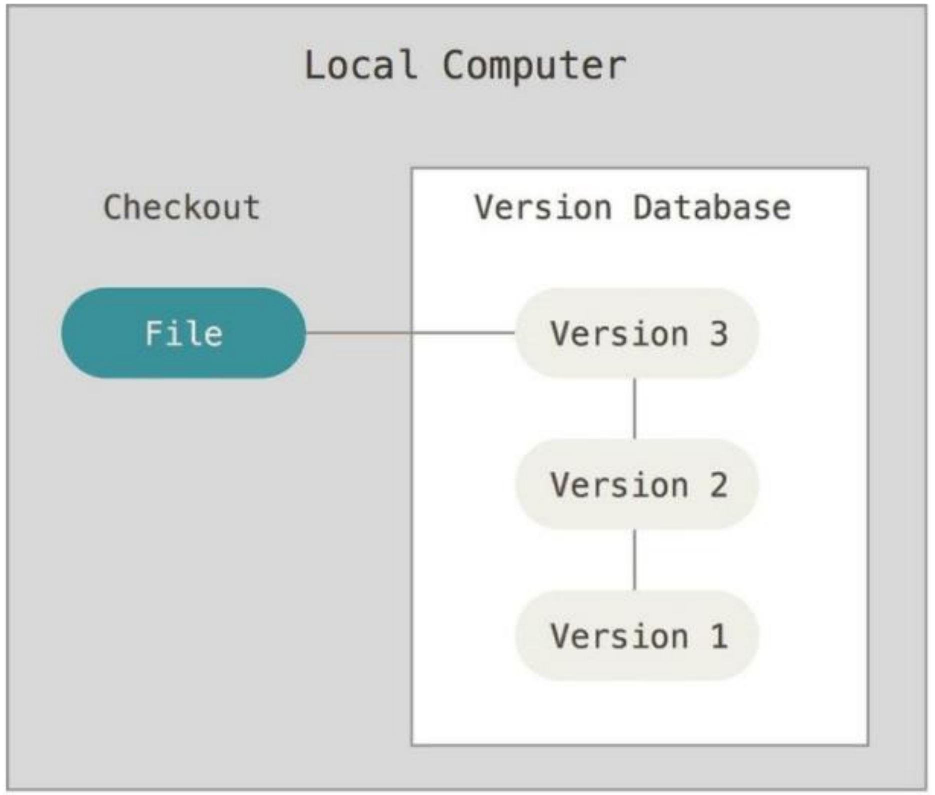
- Record **changes** to a file or a set of files (project)
- Enables you to **revert** to a previous state of a file or the entire project
- Enables you to see **who** made which changes
- Makes it easy to **experiment** on code, you can always revert to a previous, working version



LOCAL VERSION CONTROL

- **Local database** that keeps all the changes to files or projects
- Collaboration is very difficult or impossible

- E.g.: RCS

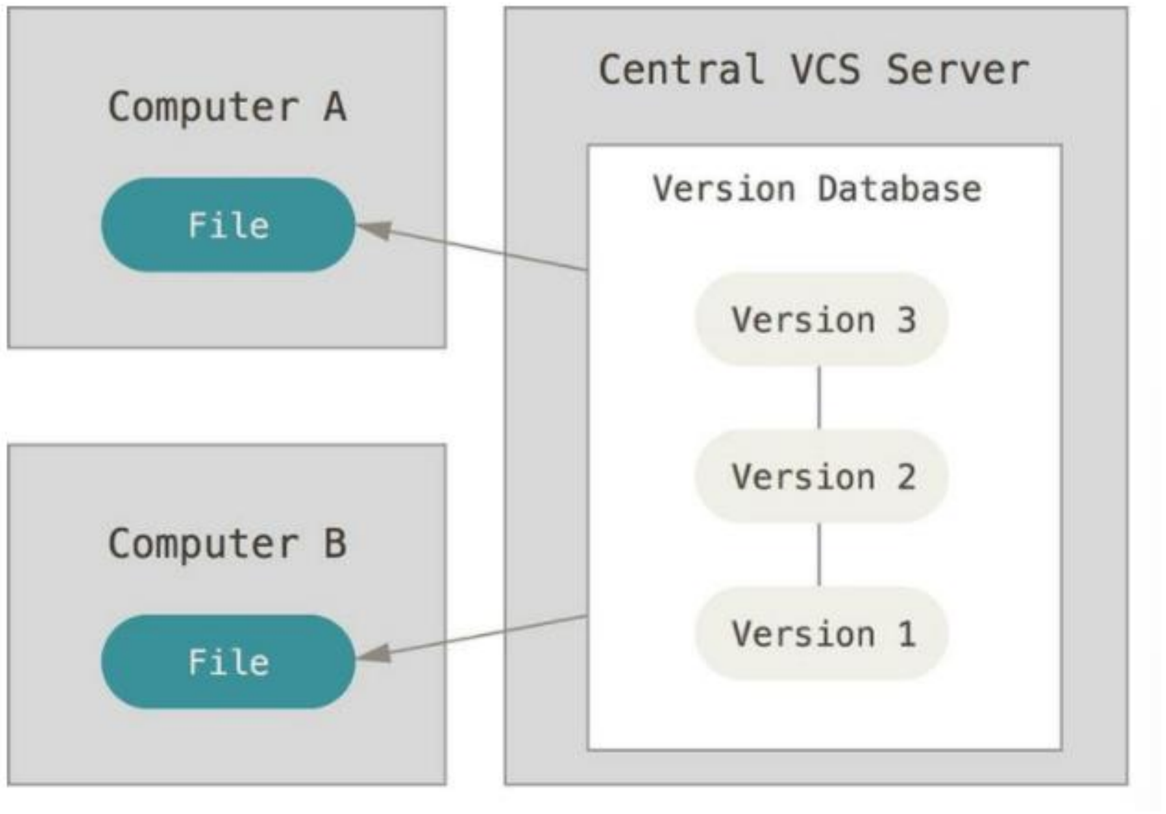


CENTRALIZED VERSION CONTROL

- **Single server** that contain all the versions of a file or project
- Clients can checkout files from the central server
- Fine grained control over who can do what

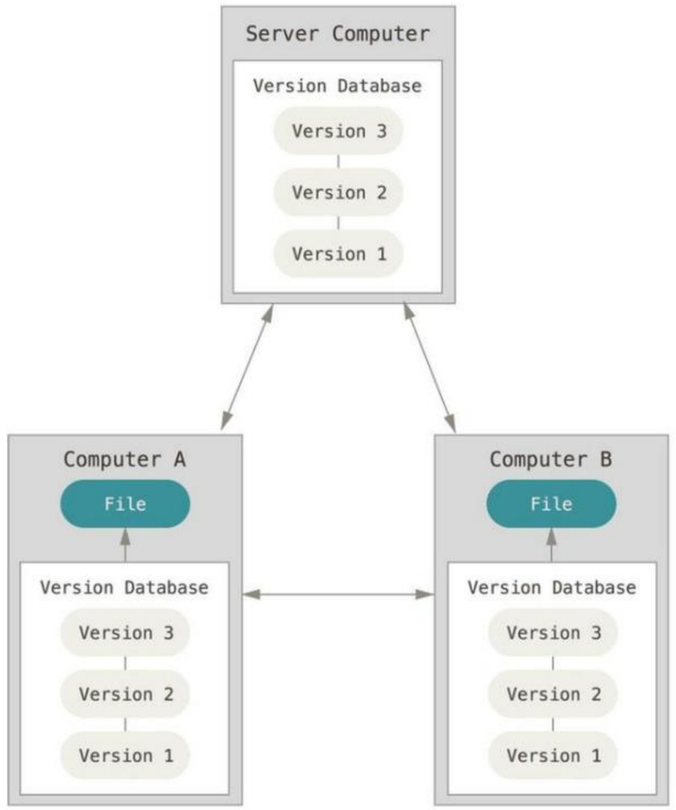
- There is a single point of failure

- E.g.: Subversion, CVS, Perforce



DISTRIBUTED VERSION CONTROL

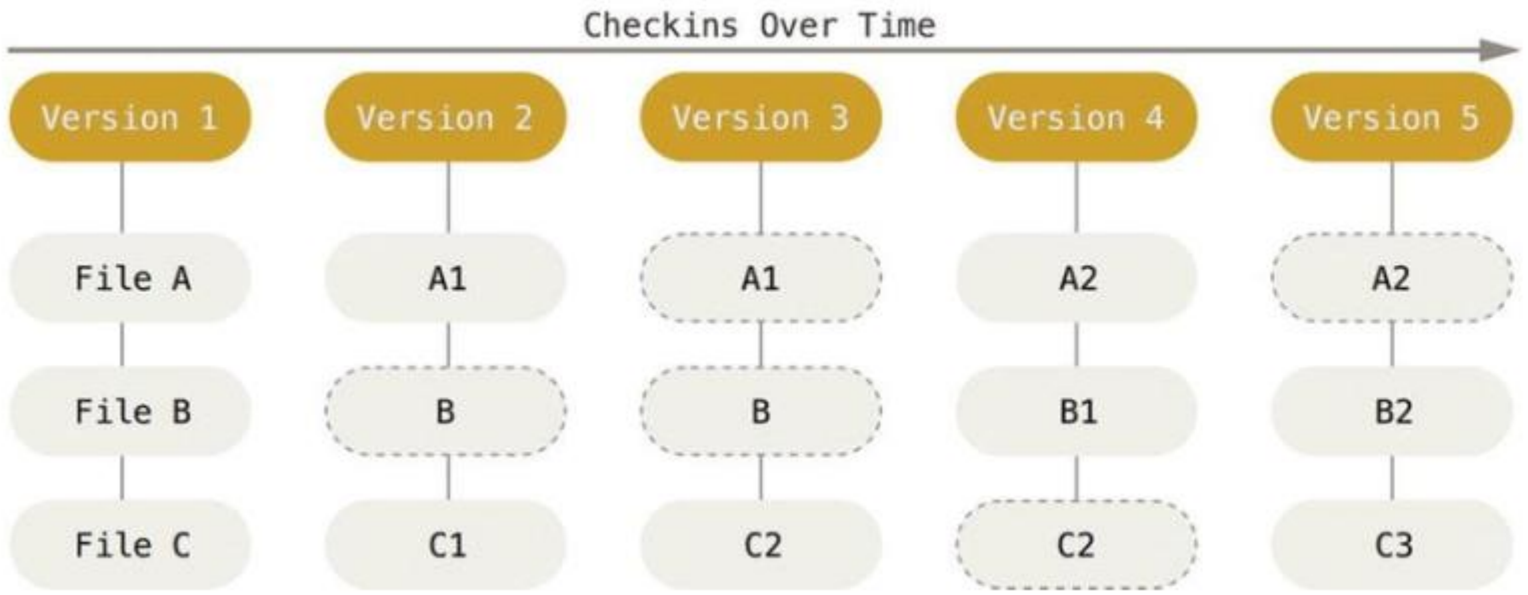
- Clients fully **mirror** (backup) the entire repository
- When the server fails, any of these copied local repositories can be placed back
- Several simultaneous servers (remotes) possible at the same time
- E.g.: Git, Mercurial, Bazaar



GIT BASICS

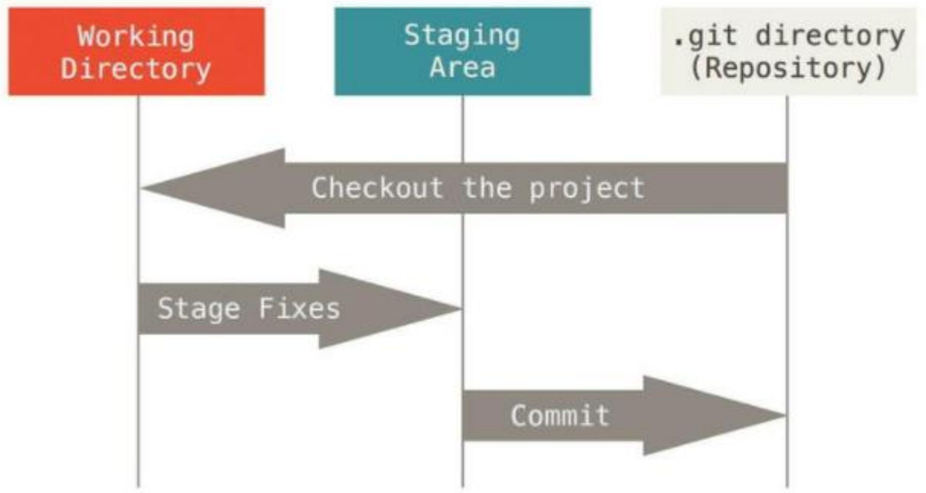
- Git takes **snapshots** of the files in a project rather than recording the changes to files as other VCS do
- If a file is not changed, the file is not stored again, only a **link to the file in the previous version**
- Most operations are **local operations**. No connection to the server is required (e.g., for commits or branching)
- Git **only adds data**, once you commit changes, it is very hard to remove or lose data.

GIT CHECK-INS OVER TIME



THE THREE STAGES

- **Committed**: Data is safely stored in your local database
- **Modified**: Files are changed but not yet committed into the local database
- **Staged**: Changed files that are marked to be committed in the next commit



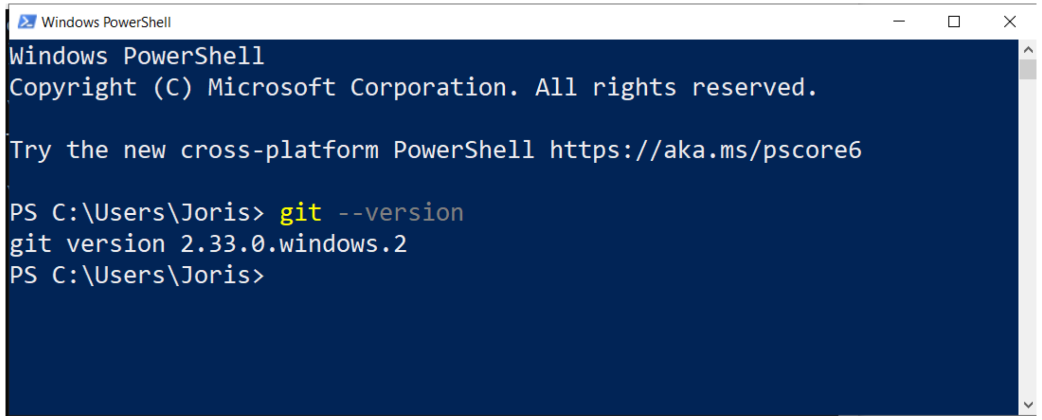
WORKFLOW

- You **modify** files in your working directory
- You **stage** the files, adding snapshots of them to your staging area
- You **commit** the files which copies the snapshots from the staging area to the local database

INSTALLING GIT

- Git command line

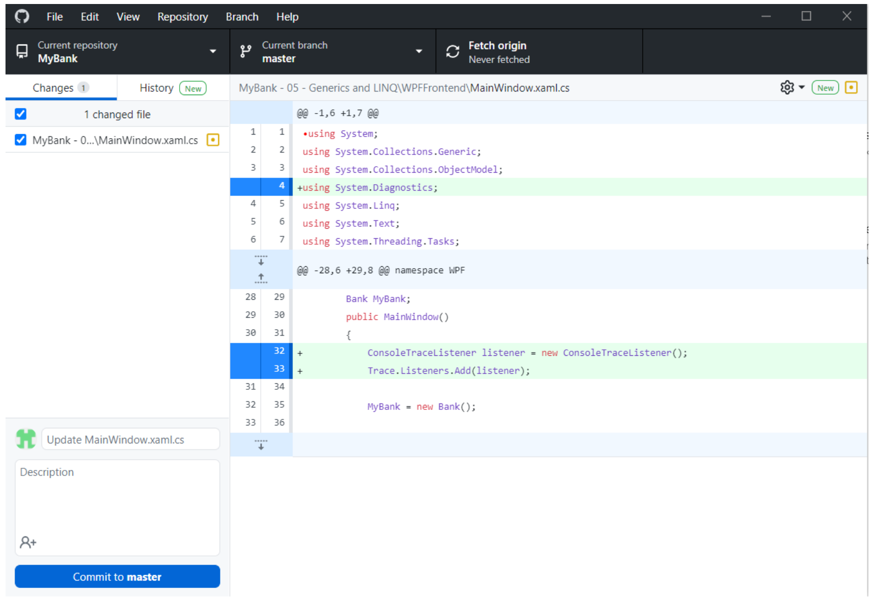
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Joris> git --version
git version 2.33.0.windows.2
PS C:\Users\Joris>
```

- Github desktop

- https://desktop.github.com/



```
@@ -1,6 +1,7 @@
 using System;
 using System.Collections.Generic;
 using System.Collections.ObjectModel;
+using System.Diagnostics;
 using System.Linq;
 using System.Text;
 using System.Threading.Tasks;
@@ -28,6 +29,8 @@ namespace WPF
         Bank MyBank;
         public MainWindow()
         {
+            ConsoleTraceListener listener = new ConsoleTraceListener();
+            Trace.Listeners.Add(listener);

             MyBank = new Bank();
```

FIRST-TIME SETUP

git config --global user.name “yourname”
git config --global user.email youremail@mail.com

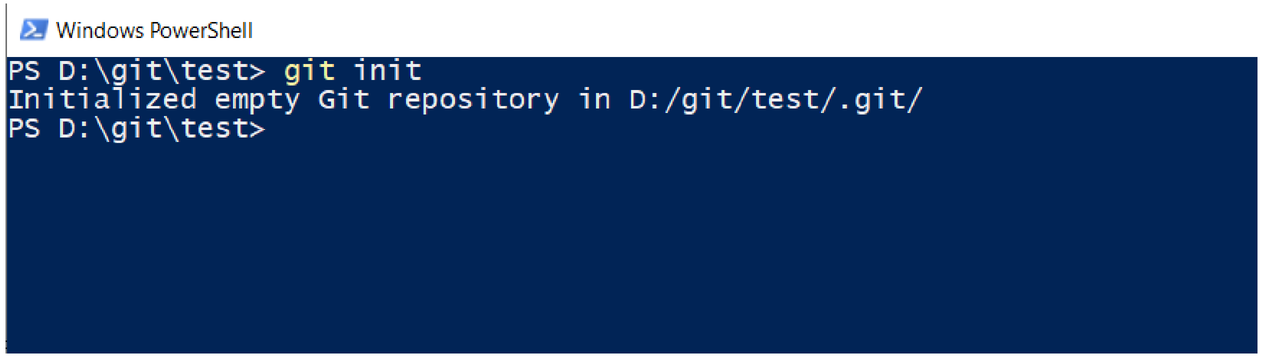
- Stores your settings in
  - $HOME\.gitconfig
  - /etc/gitconfig

GIT BASICS

STARTING A NEW PROJECT

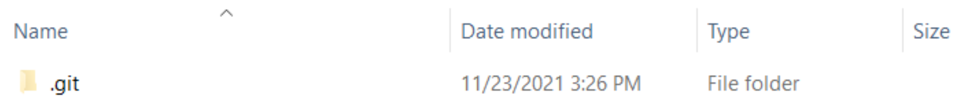
git init

- Initialize a repository in an existing directory (existing project or empty folder)
- Creates subdirectory named .git containing all necessary repository files



Windows PowerShell

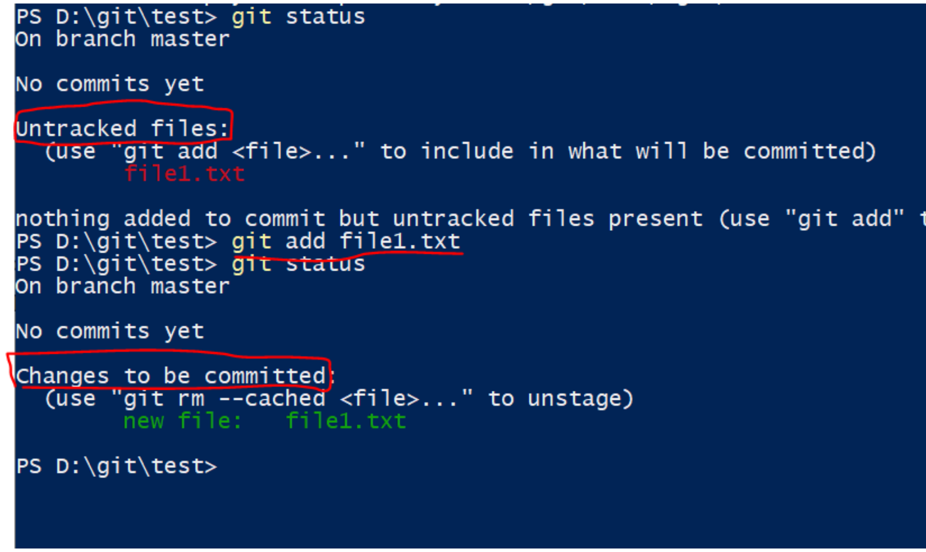
```
PS D:\git\test> git init
Initialized empty Git repository in D:/git/test/.git/
PS D:\git\test>
```



Name	Date modified	Type	Size
.git	11/23/2021 3:26 PM	File folder	

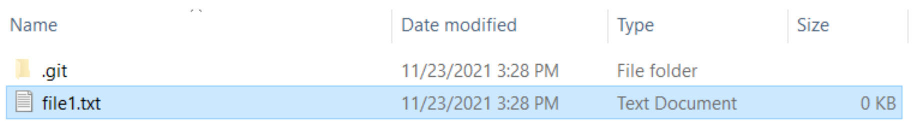
ADDING FILES TO THE STAGED AREA

git add file.c	*add file.c*
*git add *.c*	*add all .c files*
*git add folder/**	*add all files in folder*
git add –A	*add all files in this directory including subdirectories*



```
PS D:\git\test> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

nothing added to commit but untracked files present (use "git add" t
PS D:\git\test> git add file1.txt
PS D:\git\test> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

PS D:\git\test>
```

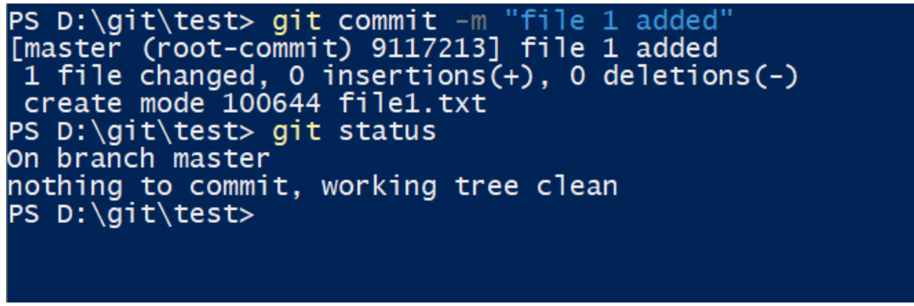
- These commands add changed files to the staging area



Name	Date modified	Type	Size
.git	11/23/2021 3:28 PM	File folder	
file1.txt	11/23/2021 3:28 PM	Text Document	0 KB

COMMITTING STAGED FILES

git commit –m “Commit message”



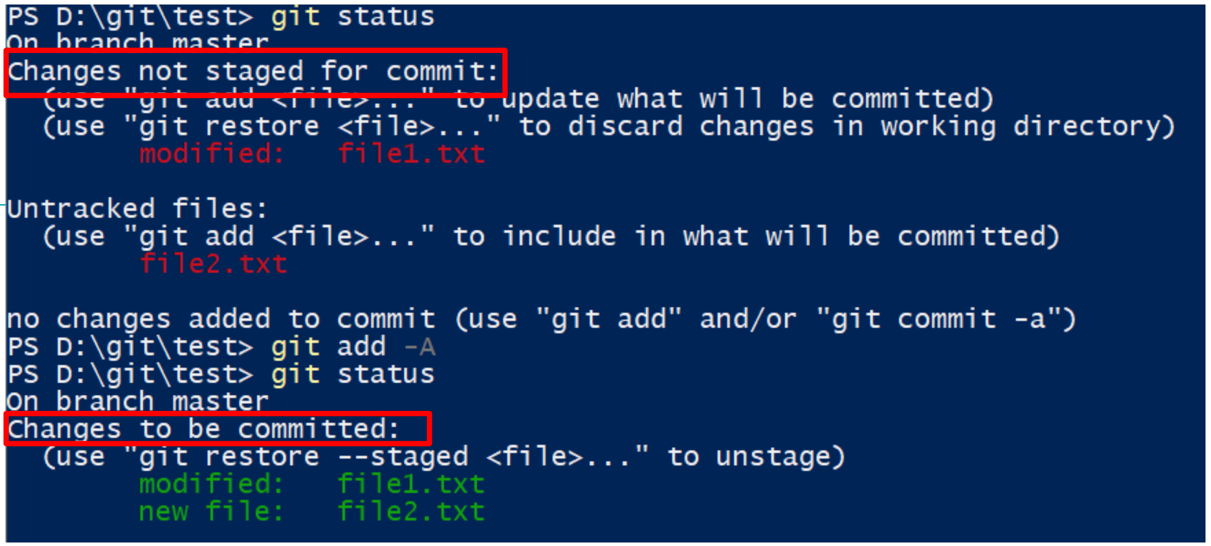
```
PS D:\git\test> git commit -m "file 1 added"
[master (root-commit) 9117213] file 1 added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
PS D:\git\test> git status
On branch master
nothing to commit, working tree clean
PS D:\git\test>
```

- Commits all staged files
- Always add a clear commit message
  - Issue 34 fixed
  - Background color changed
  - Submit button added
  - …

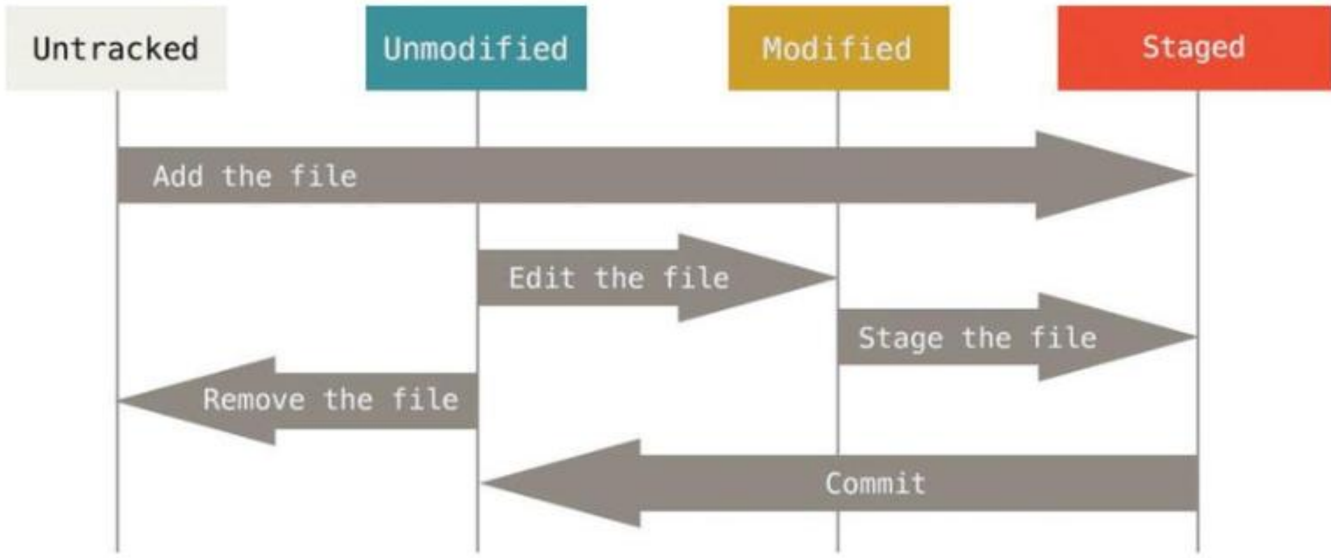
GET STATUS

git status

- Get the current status of all files in your directory
  - Untracked
  - Modified
  - New file



```
PS D:\git\test> git status
On branch 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:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS D:\git\test> git add -A
PS D:\git\test> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1.txt
        new file:   file2.txt
```



.GITIGNORE FILE

- List files that the

  git add -A

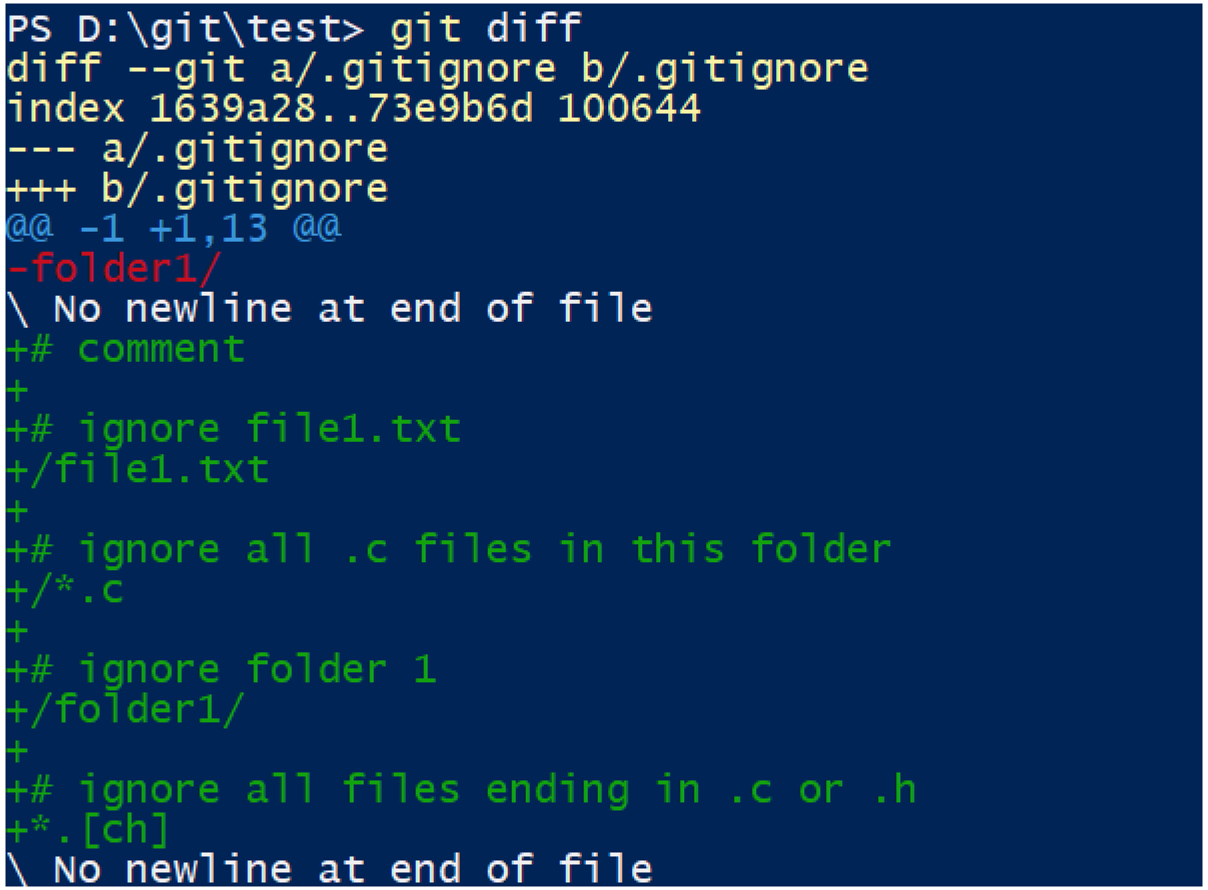
  command does not add to staging area

- Example .gitignore files: https://github.com/github/gitignore

```
# comment

# ignore file1.txt
file1.txt

# ignore all .c files in this folder
*.c

# ignore folder 1
folder1/

# ignore all files ending in .c or .h
*.[ch]

# ignore .vscode folder in any subfolder
**/.vscode/
```

SEE DIFFERENCES IN ALL FILES

git diff



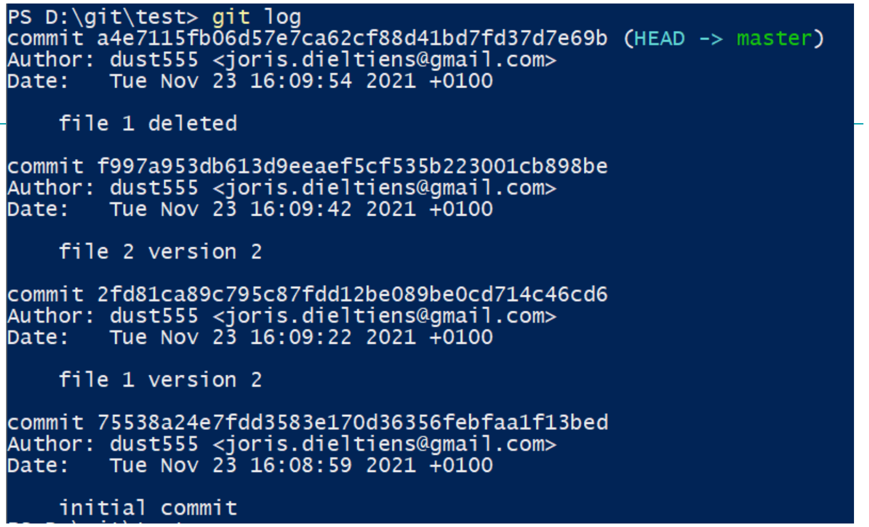
```
PS D:\git\test> git diff
diff --git a/.gitignore b/.gitignore
index 1639a28..73e9b6d 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1 +1,13 @@
-folder1/
\ No newline at end of file
+# comment
+
+# ignore file1.txt
+/file1.txt
+
+# ignore all .c files in this folder
+/*.c
+
+# ignore folder 1
+/folder1/
+
+# ignore all files ending in .c or .h
+*.[ch]
\ No newline at end of file
```

VIEWING COMMIT HISTORY

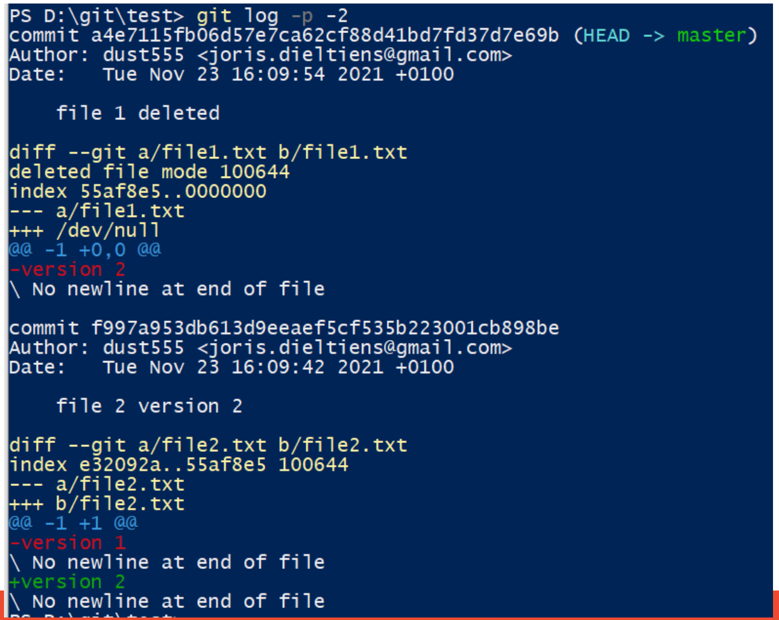
git log

git log –p -2

- show changes in last 2 commits



```
PS D:\git\test> git log
commit a4e7115fb06d57e7ca62cf88d41bd7fd37d7e69b (HEAD -> master)
Author: dust555 <joris.dieltiens@gmail.com>
Date:   Tue Nov 23 16:09:54 2021 +0100

    file 1 deleted

commit f997a953db613d9eeaef5cf535b223001cb898be
Author: dust555 <joris.dieltiens@gmail.com>
Date:   Tue Nov 23 16:09:42 2021 +0100

    file 2 version 2

commit 2fd81ca89c795c87fdd12be089be0cd714c46cd6
Author: dust555 <joris.dieltiens@gmail.com>
Date:   Tue Nov 23 16:09:22 2021 +0100

    file 1 version 2

commit 75538a24e7fdd3583e170d36356febfaa1f13bed
Author: dust555 <joris.dieltiens@gmail.com>
Date:   Tue Nov 23 16:08:59 2021 +0100

    initial commit
```



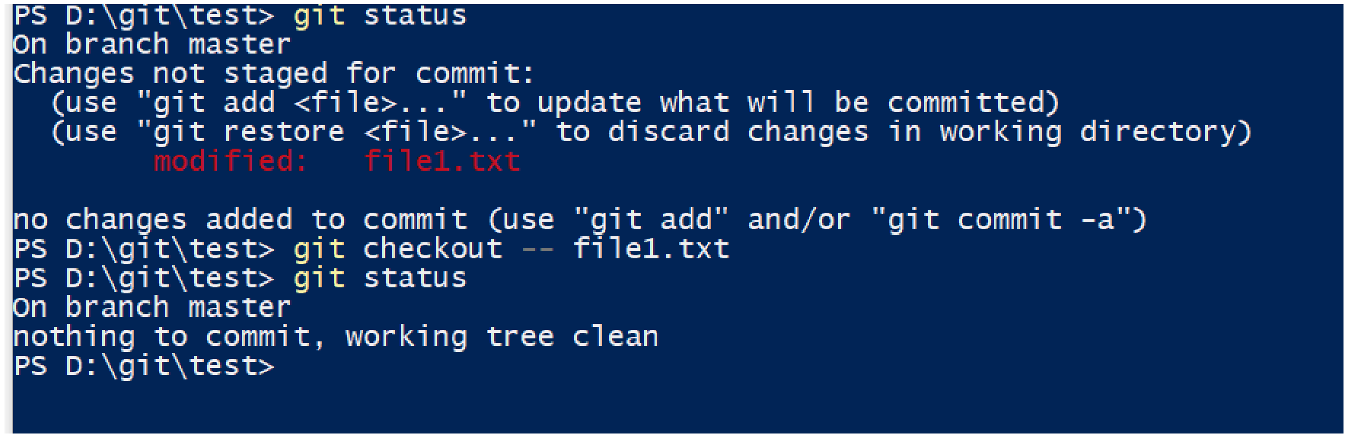
```
PS D:\git\test> git log -p -2
commit a4e7115fb06d57e7ca62cf88d41bd7fd37d7e69b (HEAD -> master)
Author: dust555 <joris.dieltiens@gmail.com>
Date:   Tue Nov 23 16:09:54 2021 +0100

    file 1 deleted

diff --git a/file1.txt b/file1.txt
deleted file mode 100644
index 55af8e5..0000000
--- a/file1.txt
+++ /dev/null
@@ -1 +0,0 @@
-version 2
\ No newline at end of file

commit f997a953db613d9eeaef5cf535b223001cb898be
Author: dust555 <joris.dieltiens@gmail.com>
Date:   Tue Nov 23 16:09:42 2021 +0100

    file 2 version 2

diff --git a/file2.txt b/file2.txt
index e32092a..55af8e5 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1 +1 @@
-version 1
\ No newline at end of file
+version 2
\ No newline at end of file
```

UNMODIFY A MODIFIED FILE

git checkout -- <filename>

Reverting to the previous version of a single file

Every change that you made to the file is gone!



```
PS D:\git\test> git status
On branch 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:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS D:\git\test> git checkout -- file1.txt
PS D:\git\test> git status
On branch master
nothing to commit, working tree clean
PS D:\git\test>
```

STARTING FROM AN EXISTING REPOSITORY

```
git clone <url>
```

- Copies a remote repository into a local repository
  - All files
  - All versions

WORKING WITH REMOTE REPOSITORY

- Collaborating on the same project
- Backup
- You can have several remotes
- **Push and pull** data to and from them

- E.g.: github

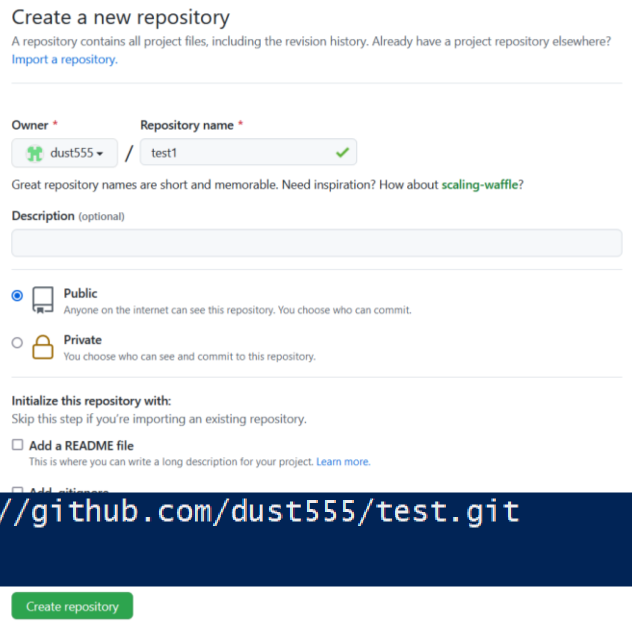
ADDING A REMOTE TO YOUR LOCAL REPO

git remote add <ShortName> <url>

- Add a remote repository

git remote –v

- List all remotes



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * dust555 / Repository name * test1 ✓

Great repository names are short and memorable. Need inspiration? How about scaling-waffle?

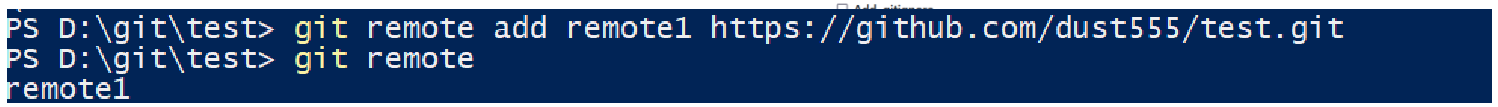
Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

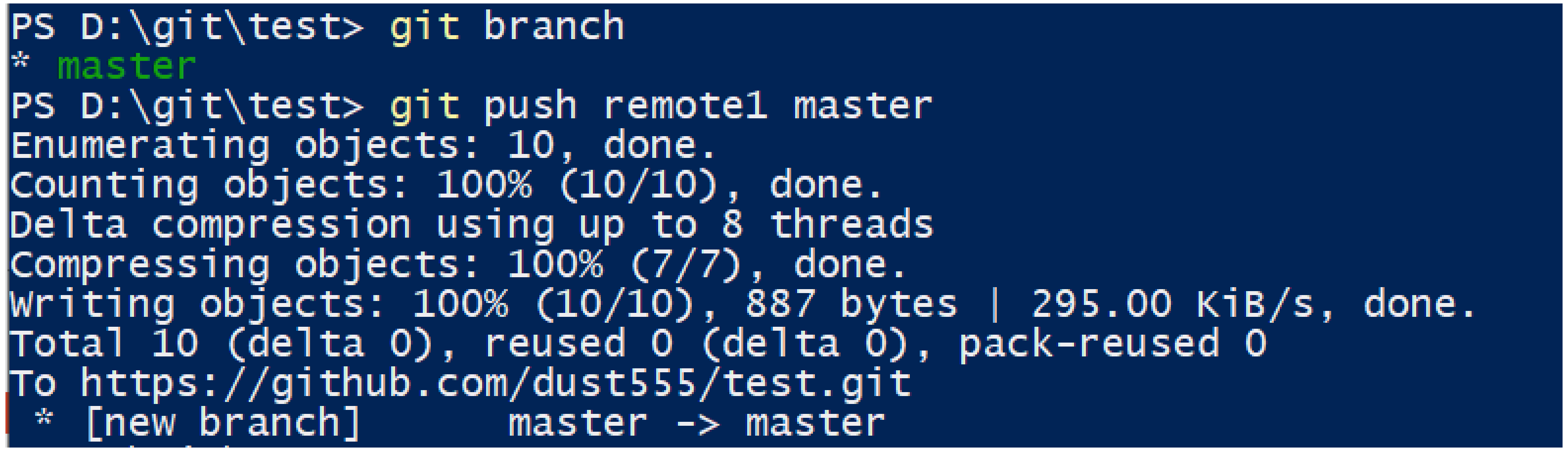
☐ Add a README file
This is where you can write a long description for your project. Learn more.



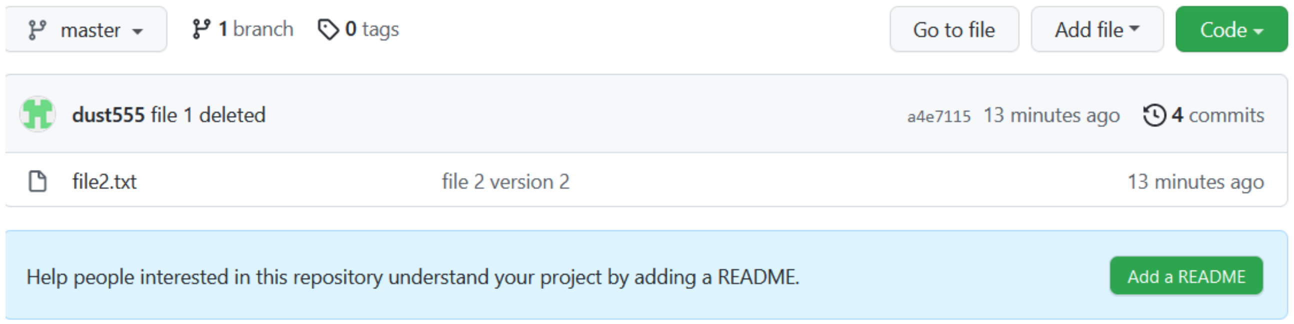
```
PS D:\git\test> git remote add remote1 https://github.com/dust555/test.git
PS D:\git\test> git remote
remote1
```

PUSHING YOUR DATA TO YOUR REMOTE

git push <RemoteName> <BranchName>



```
PS D:\git\test> git branch
* master
PS D:\git\test> git push remote1 master
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 887 bytes | 295.00 KiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/dust555/test.git
 * [new branch]      master -> master
```



dust555 file 1 deleted a4e7115 13 minutes ago 4 commits

file2.txt file 2 version 2 13 minutes ago

Help people interested in this repository understand your project by adding a README.

PULLING FROM A REMOTE

git fetch <RemoteName>

- Get all the information about the remote repository without changing local files

git pull <RemoteName> <BanchName>

- Get all the data and integrate this in your local repository

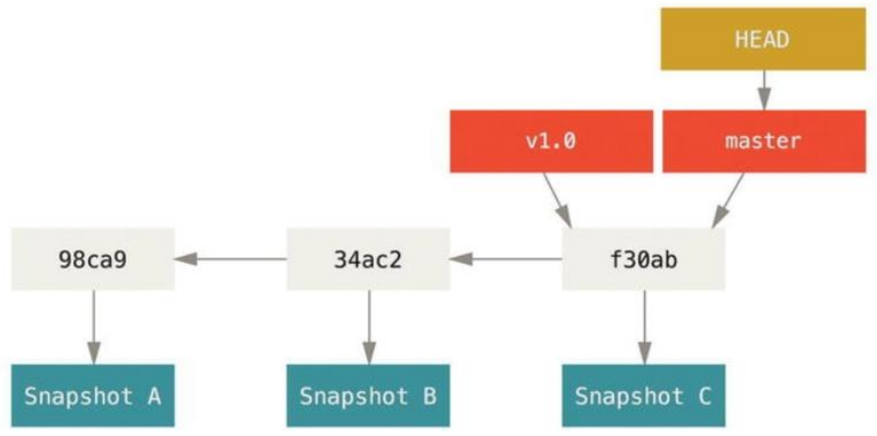
RENAME AND REMOVE REMOTE

```
git remote rename <From> <To>

git remote rm <RemoteName>
```

BRANCHING

WHAT IS BRANCHING

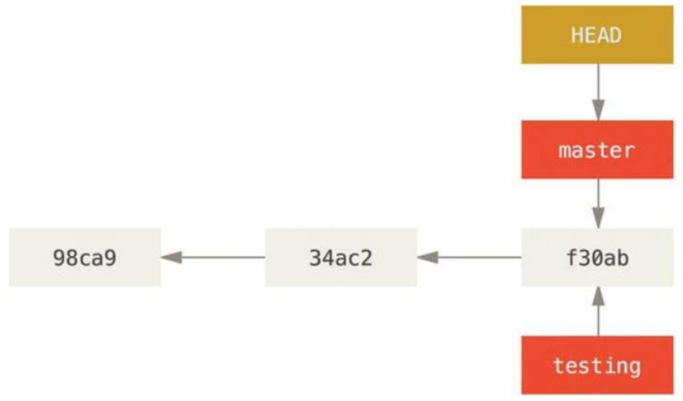
- Diverge from main line and continue developing without messing with the main line
- Git branches are lightweight
- Commits are pointers to snapshots of your work
- Default pointer is "master" created with *git init*
- A branch is an additional pointer to one of the snapshots
- HEAD marks the pointer to the current state of the directory



CREATING A BRANCH

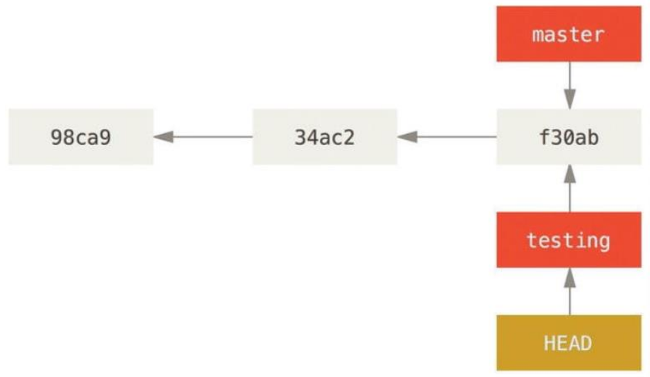
git branch testing

- Create a branch called testing



git checkout testing

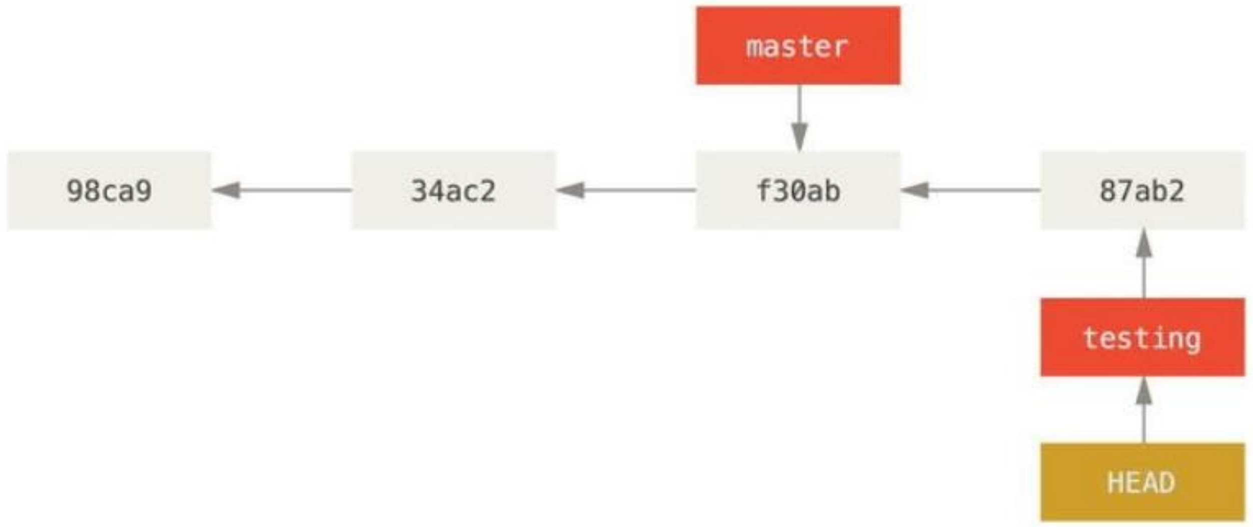
- Move your head to a different branch



COMMITTING IN A BRANCH

git commit –a –m “change to testing”

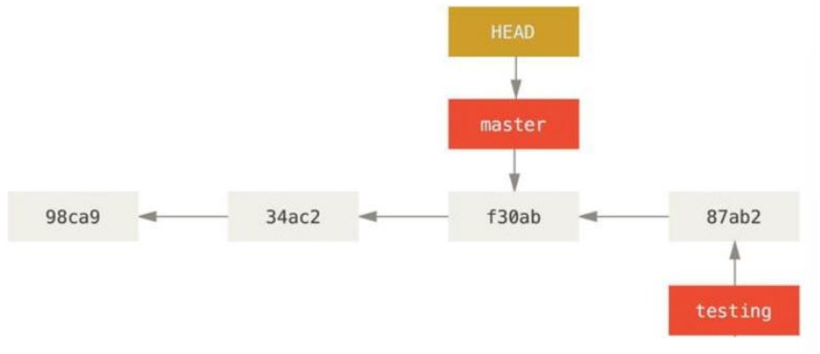
- Make a new snapshot and move the current branch pointer to the new snapshot



MOVING BETWEEN BRANCHES

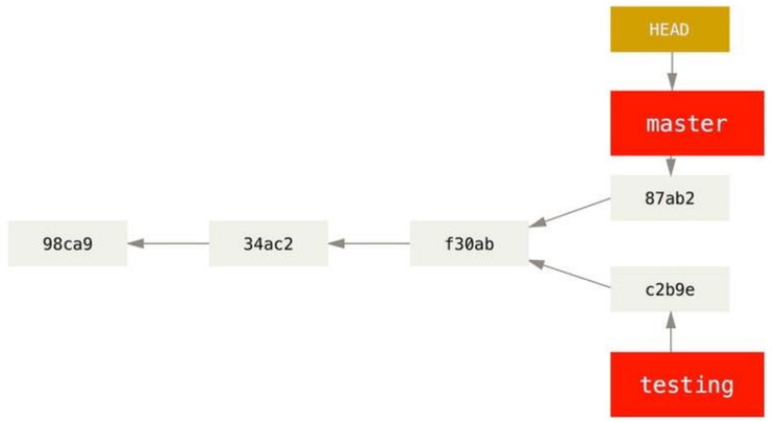
git checkout master

- Move HEAD pointer to master pointer. **The content of your folder will match this snapshot**



git commit –a –m "changes to master"

- Make a new snapshot and move the master pointer to this snapshot

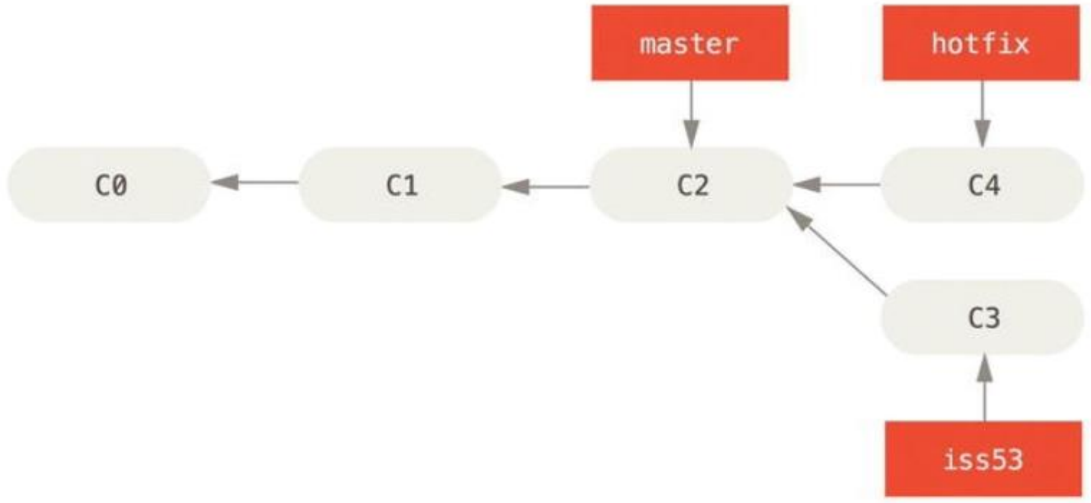


WORKING WITH BRANCHES

1. Do work on a web site.
2. Create a branch for a new story you're working on.
3. Do some work in that branch.
4. At this stage, you'll receive a call that another issue is critical and you need a hotfix. You'll do the following:
5. Switch to your production branch.
6. Create a branch to add the hotfix.
7. After it's tested, merge the hotfix branch, and push to production.
8. Switch back to your original story and continue working.

WORKING WITH BRANCHES

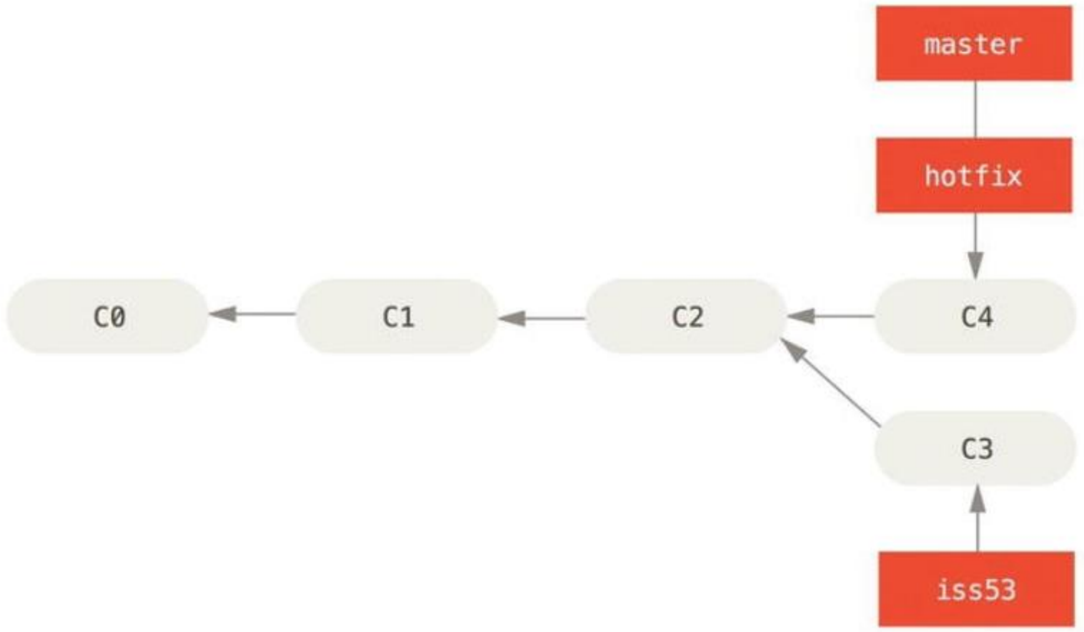
```
git branch iss53
git checkout iss53
git commit –a –m “changes”
git branch hotfix
git checkout hotfix
git commit –a –m “changes”
```



MERGING BRANCHES

- Fast forward merging

git checkout master
git merge hotfix

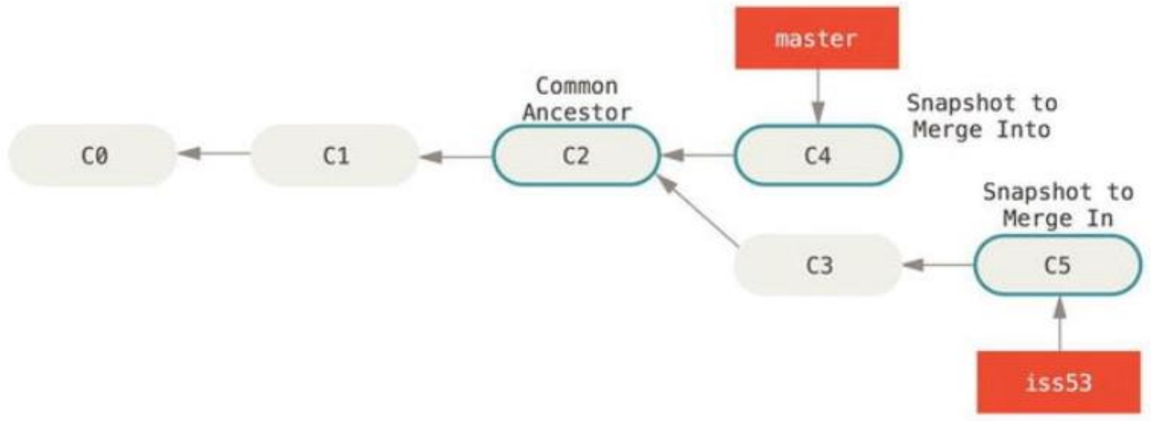


MERGING BRANCHES

- recursive merge

```
git merge iss53
```

- This will create a new snapshot which is a 3 way merge starting from a common ancestor



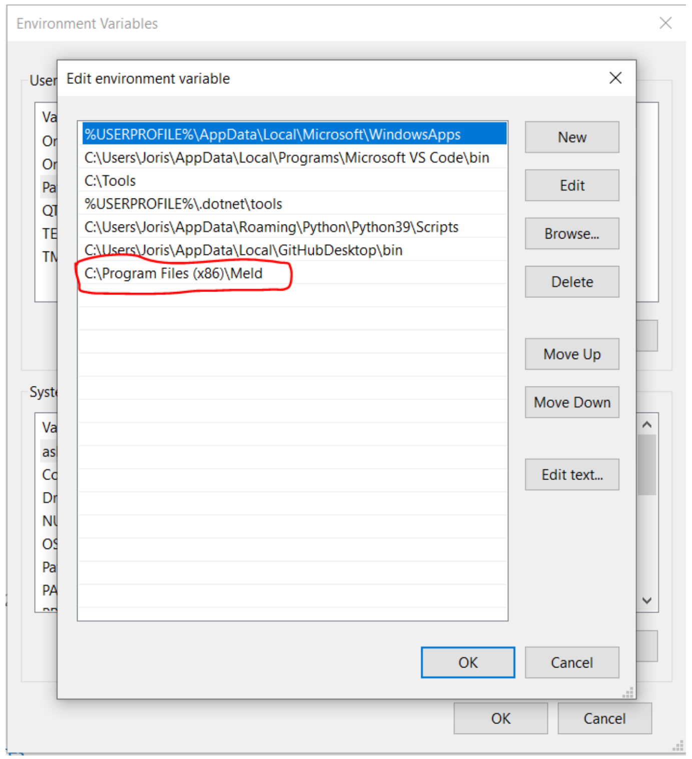
MERGE CONFLICTS

- When changing the same part of a file, merge conflicts can occur.
- No new commit is created
- You must resolve the issues by hand

MELD AS YOUR MERGE CONFLICT RESOLVER

- meld is a tool to resolve merge issues
  - Install meld
  - add meld path to the PATH environment variable
  - add the following to the .gitconfig file

```
[merge]
    tool = meld
[mergetool "meld"]
    # Choose one of these 2 lines (not both!) explained below.
    cmd = meld "$LOCAL" "$MERGED" "$REMOTE" --output "$MERGED"
    cmd = meld "$LOCAL" "$BASE" "$REMOTE" --output "$MERGED"
```



```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Joris\AppData\Local\Programs\Microsoft VS Code\bin
C:\Tools
%USERPROFILE%\.dotnet\tools
C:\Users\Joris\AppData\Roaming\Python\Python39\Scripts
C:\Users\Joris\AppData\Local\GitHubDesktop\bin
C:\Program Files (x86)\Meld
```

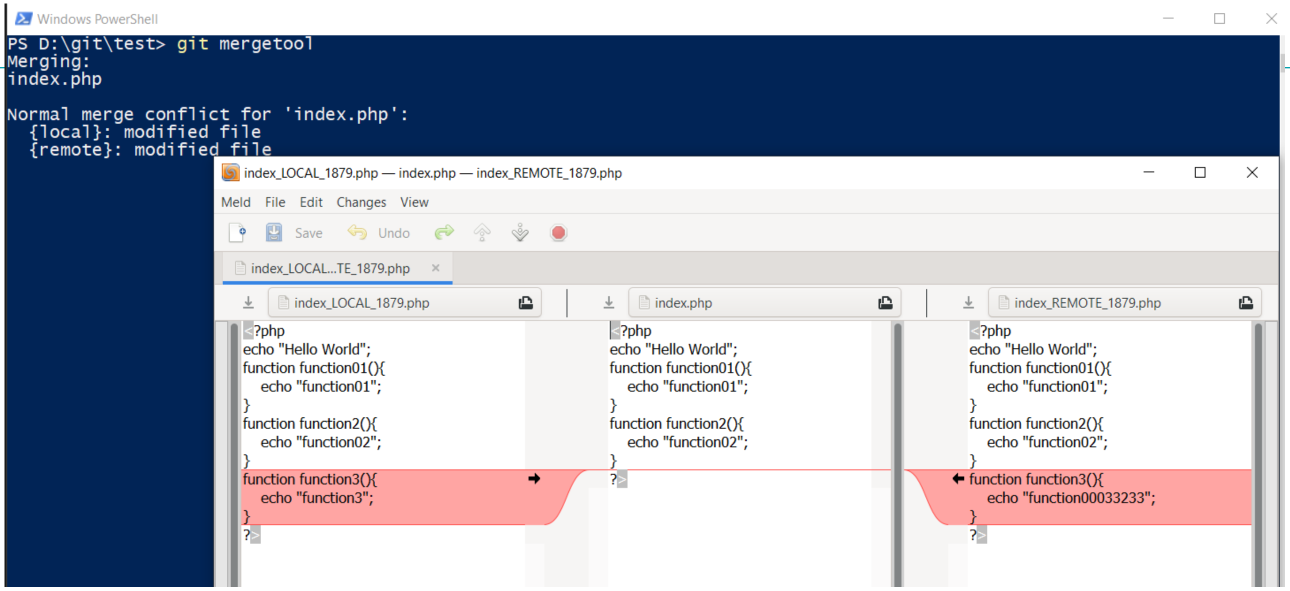
RESOLVING MERGE CONFLICTS

git mergetool

- will open meld to resolve the merge conflicts

git commit –m "merge"

- After saving the merged file, commit the changes

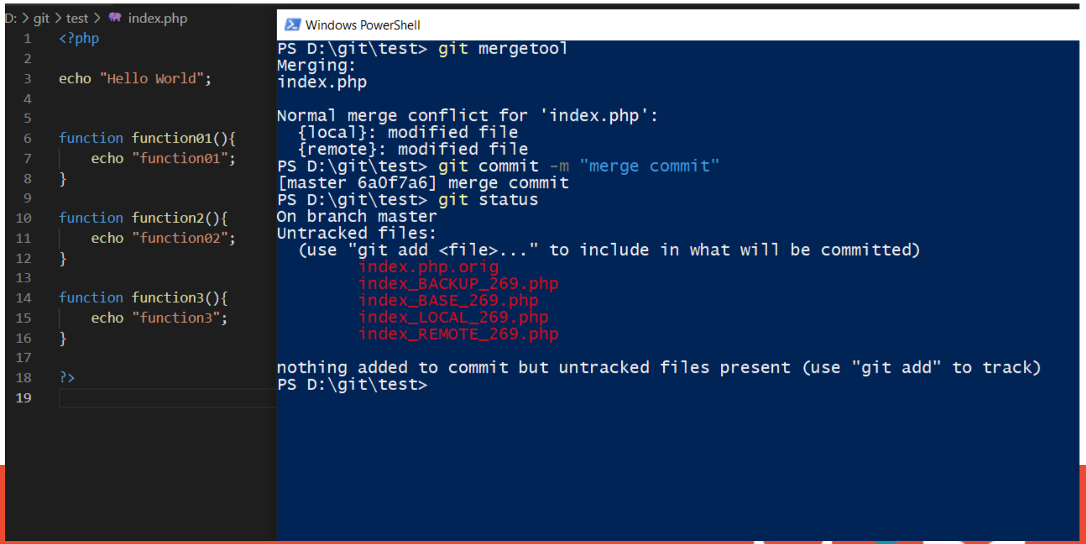

Windows PowerShell
index_LOCAL_1879.php — index.php — index_REMOTE_1879.php
Meld File Edit Changes View
Save
Undo
index_LOCAL...TE_1879.php
index_LOCAL_1879.php
index.php
index_REMOTE_1879.php

```
PS D:\git\test> git mergetool
Merging:
index.php

Normal merge conflict for 'index.php':
  {local}: modified file
  {remote}: modified file
```

```
<?php
echo "Hello World";
function function01(){
    echo "function01";
}
function function2(){
    echo "function02";
}
function function3(){
    echo "function3";
}
?>
```

```
<?php
echo "Hello World";
function function01(){
    echo "function01";
}
function function2(){
    echo "function02";
}
?>
```

```
<?php
echo "Hello World";
function function01(){
    echo "function01";
}
function function2(){
    echo "function02";
}
function function3(){
    echo "function00033233";
}
?>
```



```
<?php

echo "Hello World";


function function01(){
    echo "function01";
}

function function2(){
    echo "function02";
}

function function3(){
    echo "function3";
}

?>

```

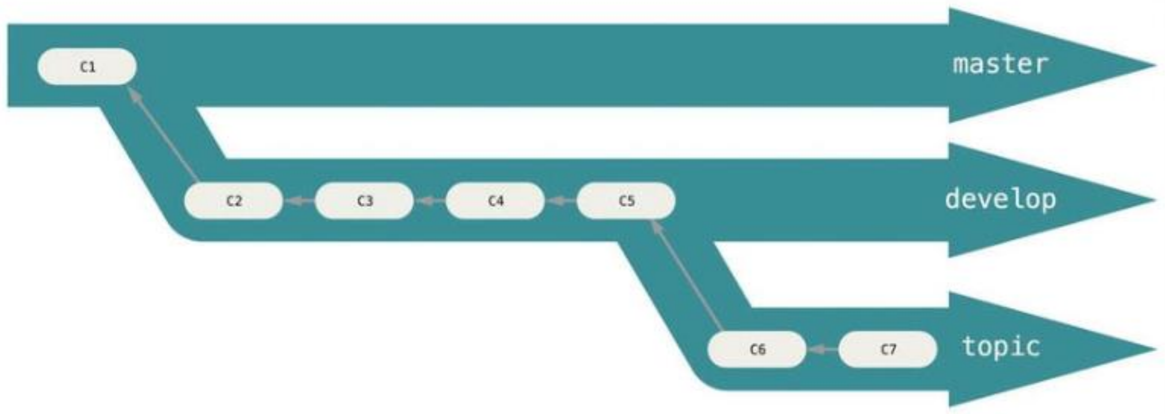
```
PS D:\git\test> git mergetool
Merging:
index.php

Normal merge conflict for 'index.php':
  {local}: modified file
  {remote}: modified file
PS D:\git\test> git commit -m "merge commit"
[master 6a0f7a6] merge commit
PS D:\git\test> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.php.orig
        index_BACKUP_269.php
        index_BASE_269.php
        index_LOCAL_269.php
        index_REMOTE_269.php

nothing added to commit but untracked files present (use "git add" to track)
PS D:\git\test>
```

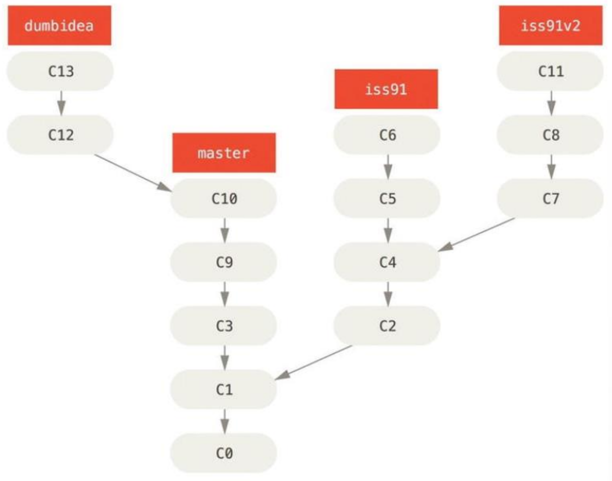
DELETE BRANCH

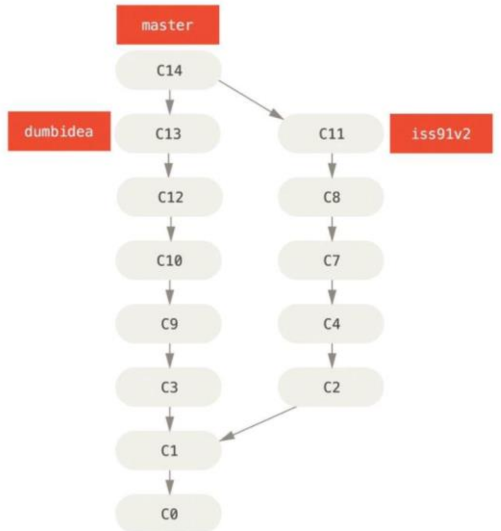
```
git branch –d <BranchName>
```

BRANCH MANAGEMENT

- **Master branch is always a stable version of your software**
- You don’t work in the master branch
- You create one or more branches to work in
- When a new feature is stable and tested, merge branch into master branch
- delete the feature branch

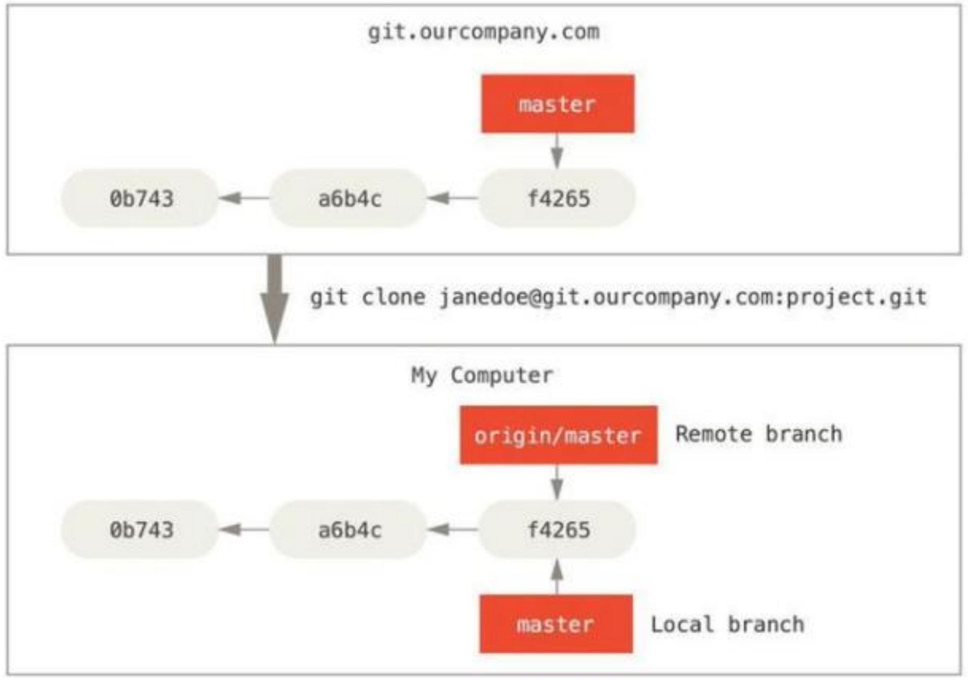


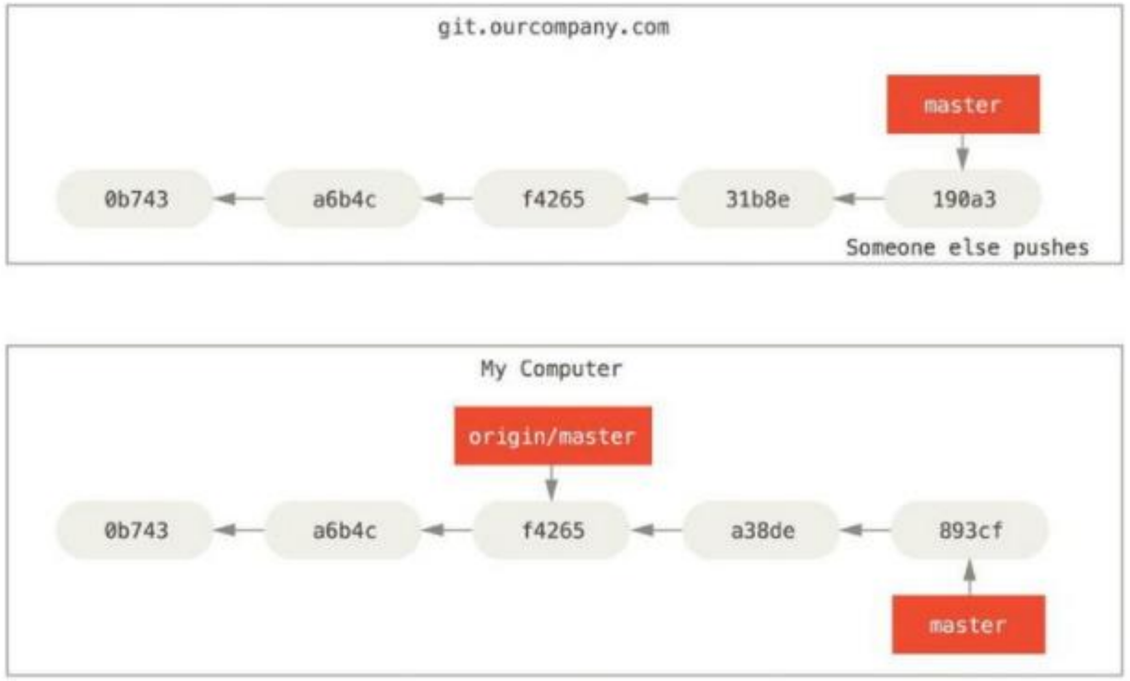




REMOTE BRANCHES

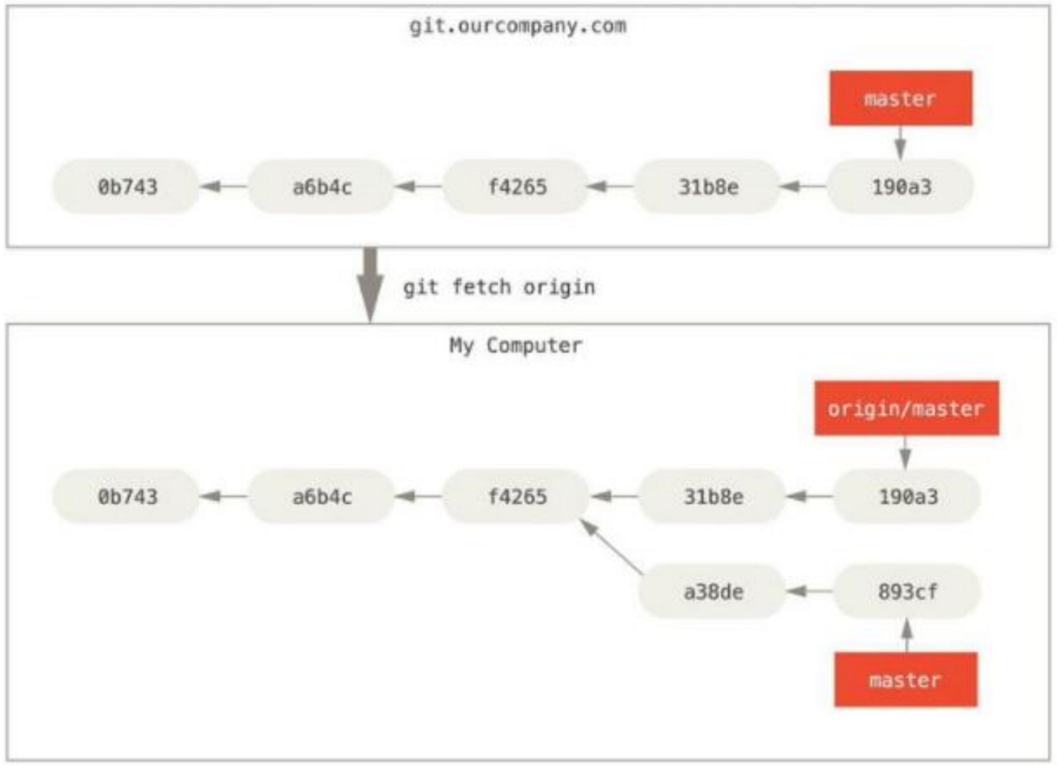
- Pointers to the state of a branch in a remote repo
- Have the form <RemoteName>/<BranchName>





FETCHING REMOTE

git fetch <RemoteName>



PUSHING TO REMOTE

- Local branches aren’t automatically synchronized to a remote

```
    git push <RemoteName> <BranchName>
```

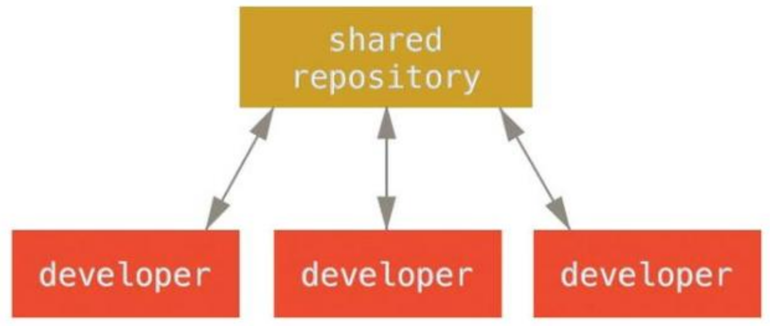
PULLING

- Fetching will fetch all the changes on the server to your local machine, but won’t modify your working directory
- You must merge yourself

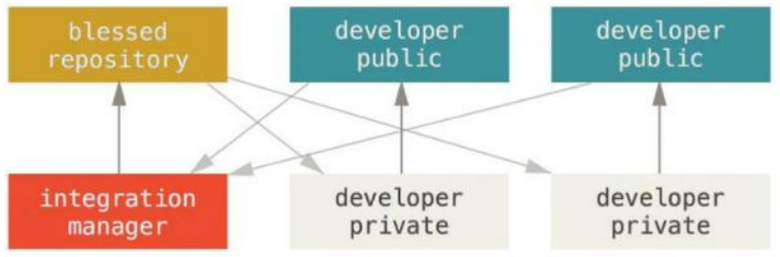
```
git pull <RemoteName> <BranchName>
```

- Merges automatically
  - = git fetch followed by a git merge

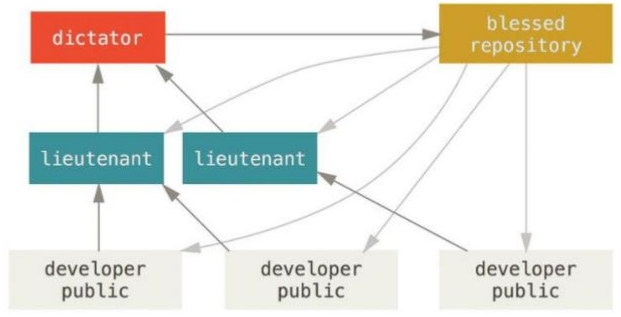
WORKING IN TEAMS



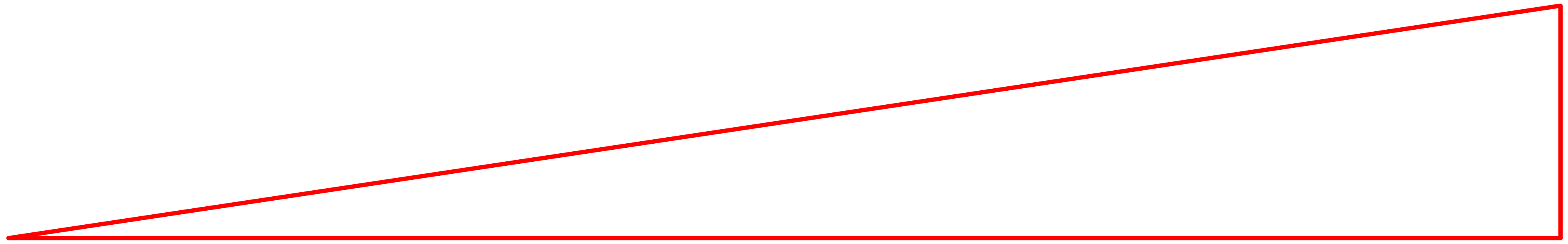
Centralized Workflow



Integration Manager

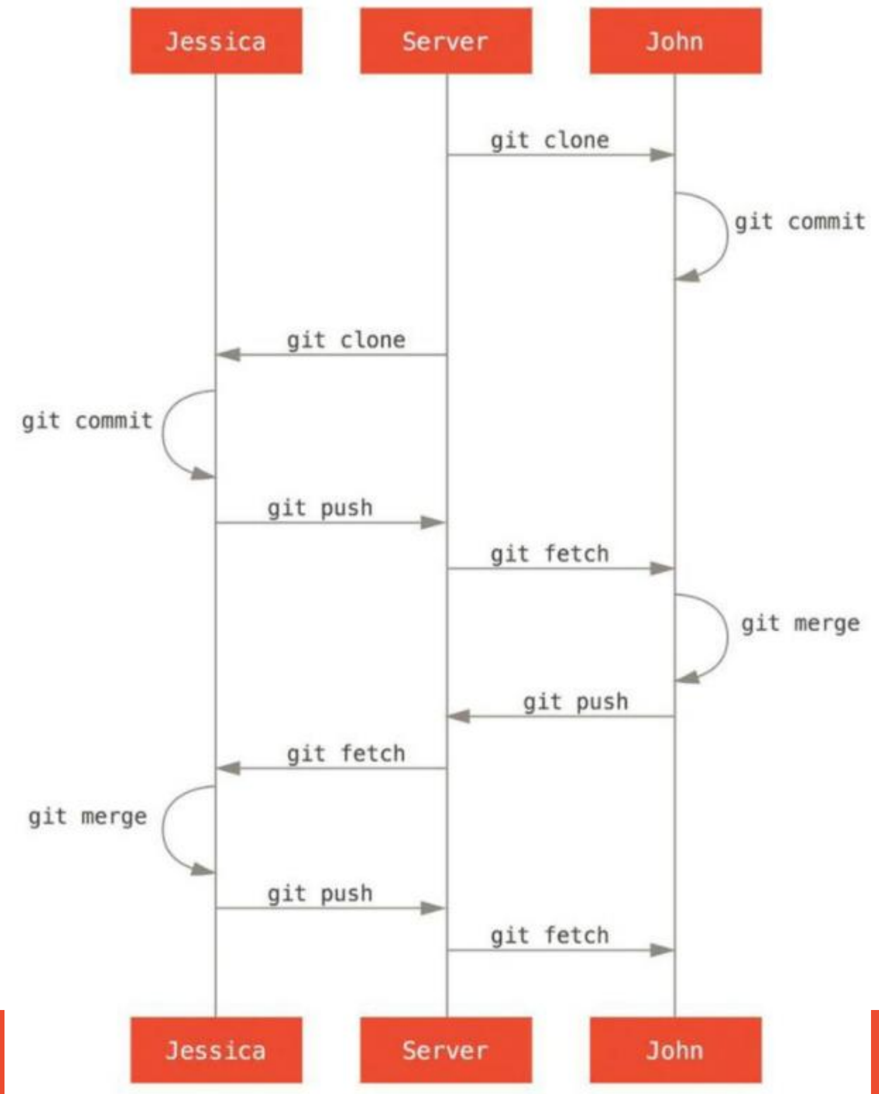


Benevolent Dictator

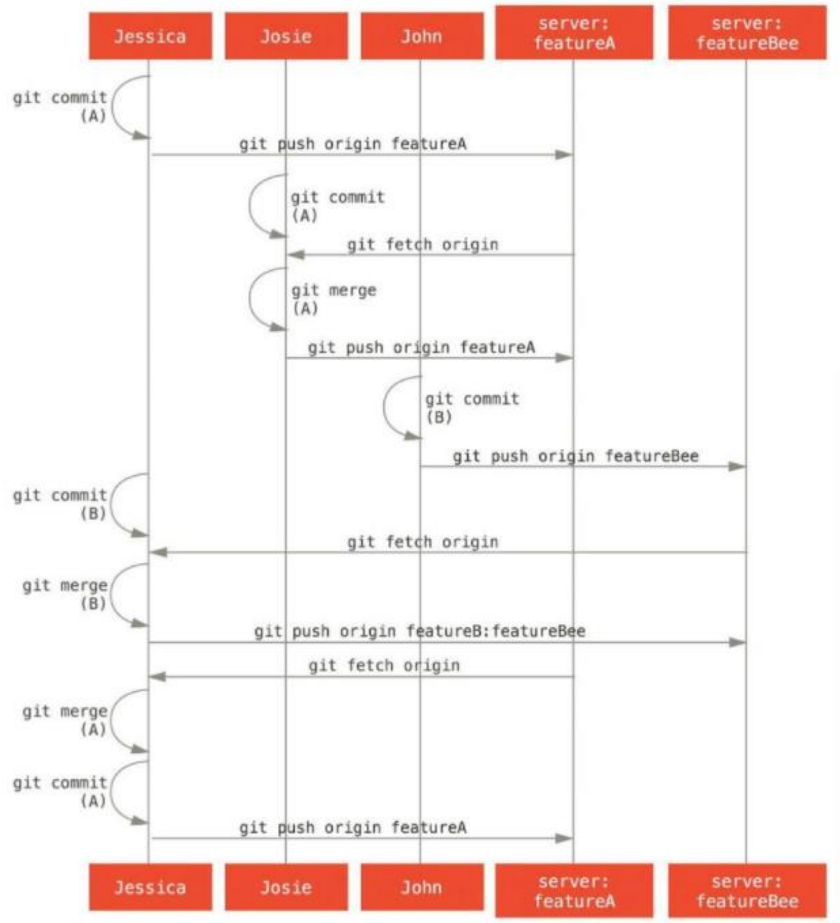


Team and project size

WORKING IN A SMALL TEAM



WORKING WITH MULTIPLE REMOTES



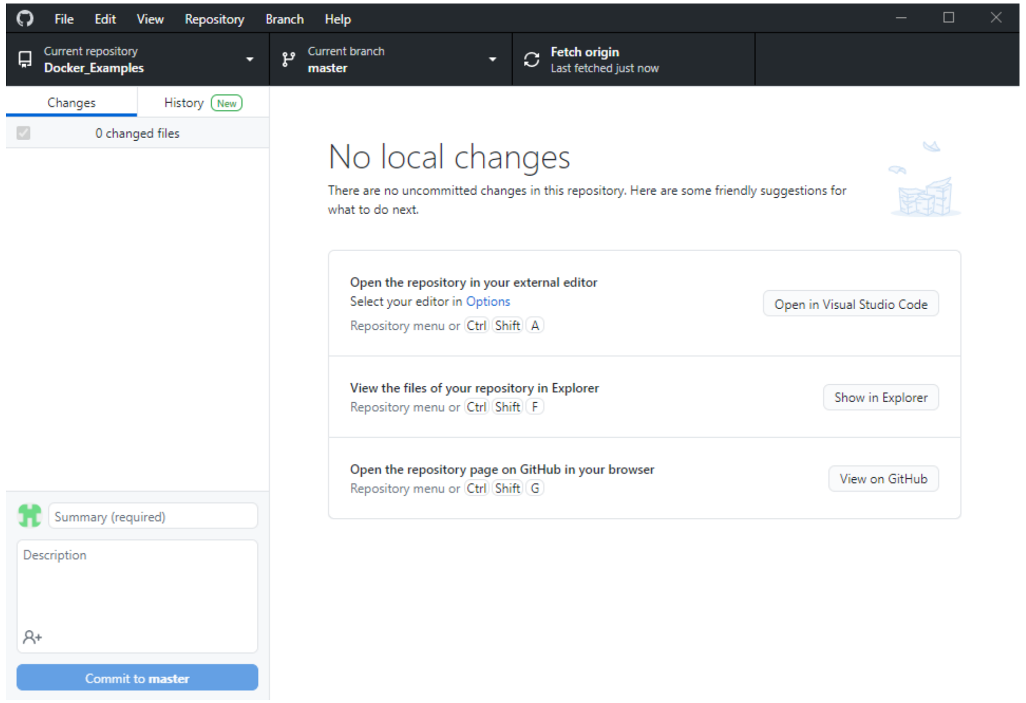
GITHUB

GITHUB

- Largest host for git repositories
- Public and private repositories
- Collaborating with multiple users

INSTALLING GITHUB DESKTOP

- Create account on github
- Download and install: https://desktop.github.com/
- GUI to do all management



No local changes

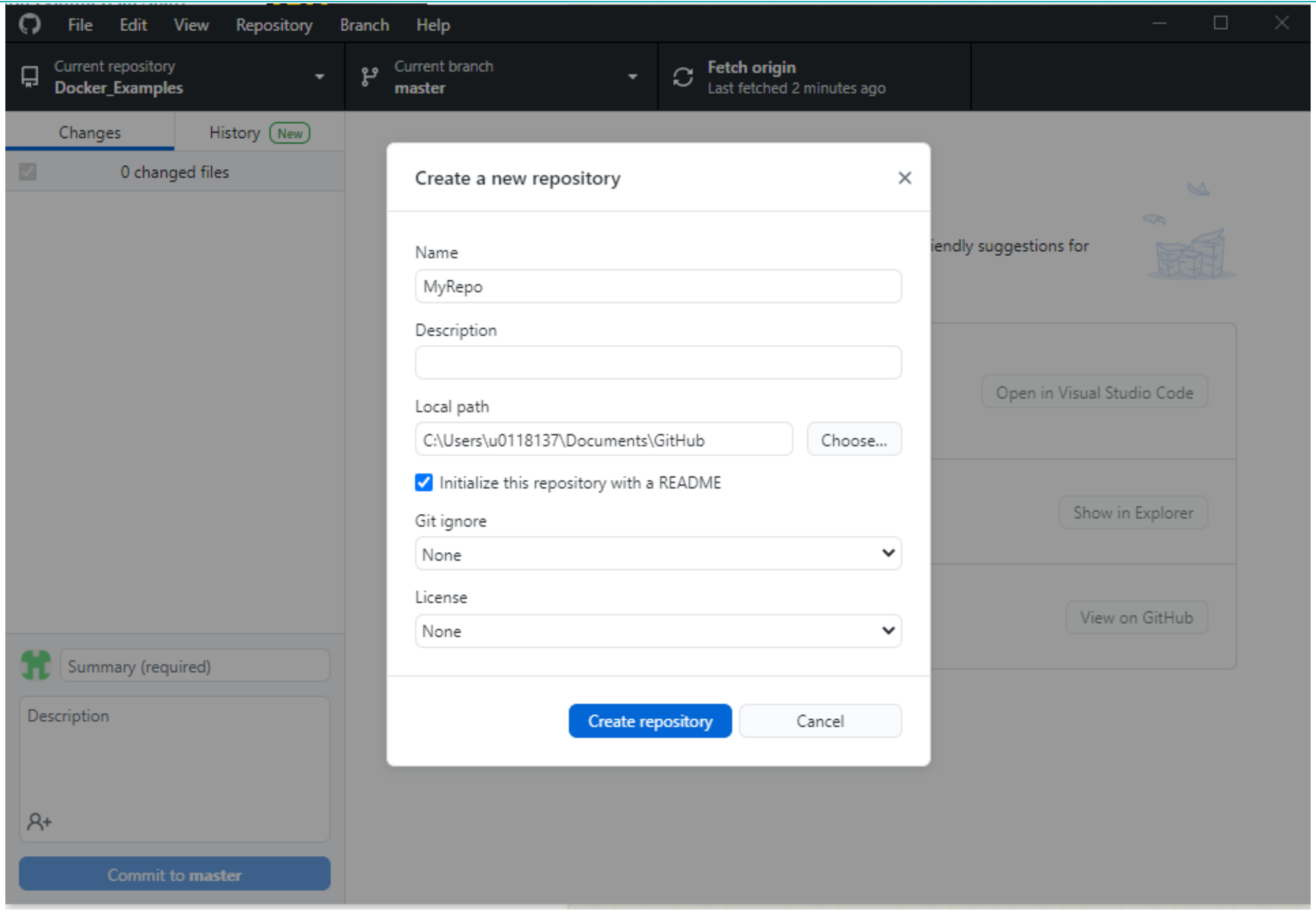
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A

View the files of your repository in Explorer
Repository menu or Ctrl Shift F

Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G

CREATING NEW REPO



Create a new repository

Name: MyRepo

Description:

Local path: C:\Users\u0118137\Documents\GitHub Choose...

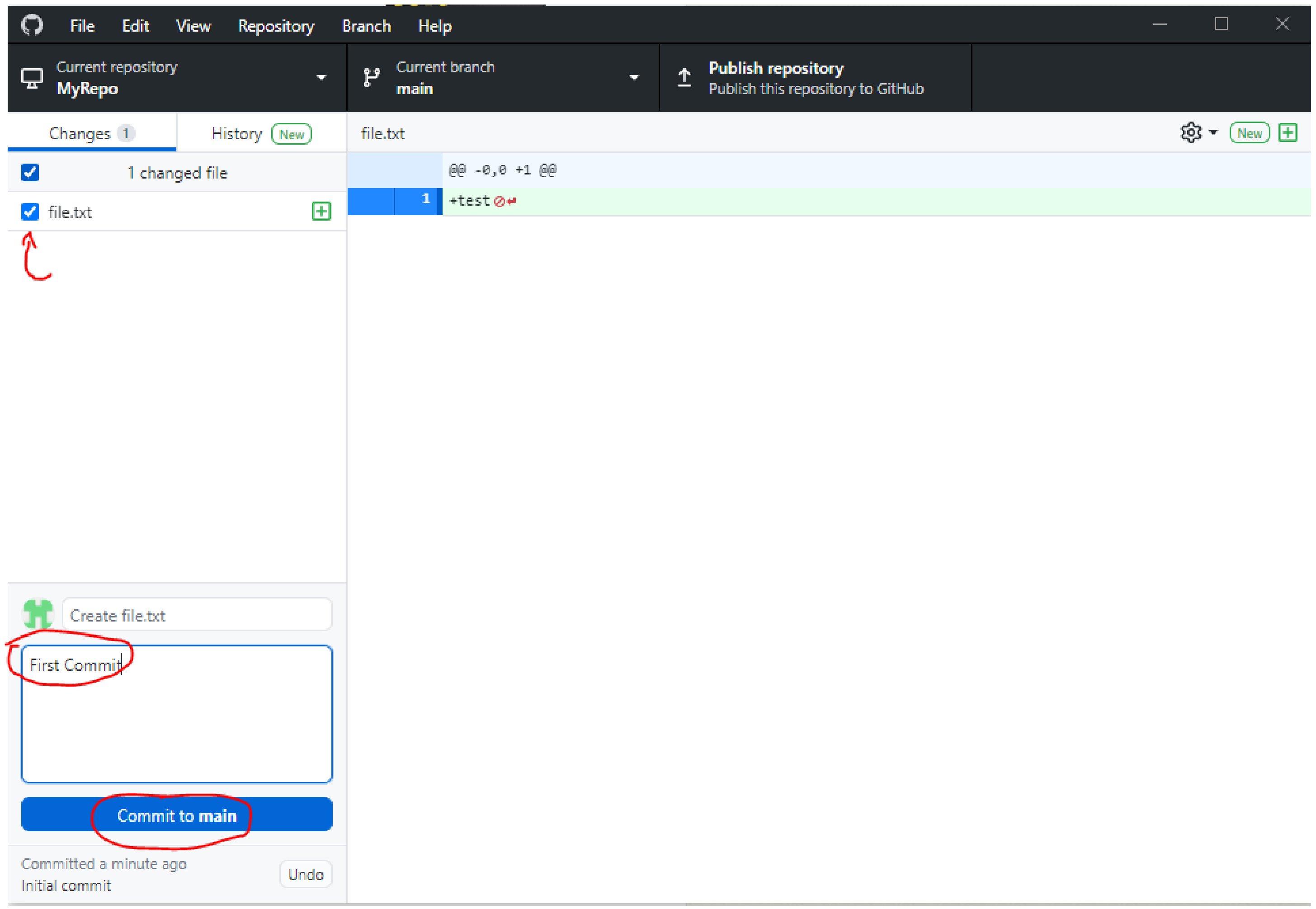
☑ Initialize this repository with a README

Git ignore: None

License: None

Create repository Cancel

COMMIT

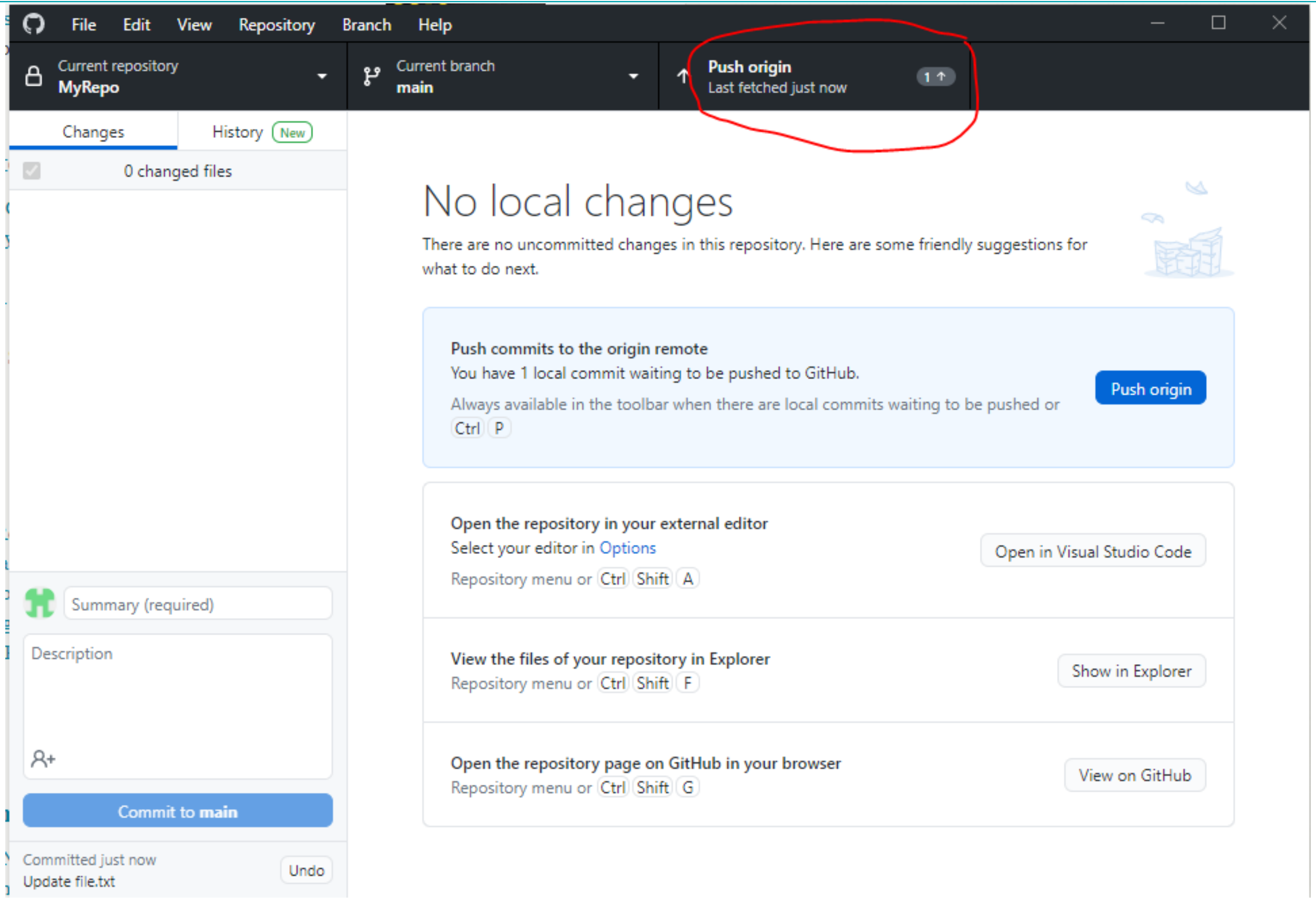


```
@@ -0,0 +1 @@
1 +test
```

First Commit

Committed a minute ago
Initial commit

PUSH TO GITHUB



No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

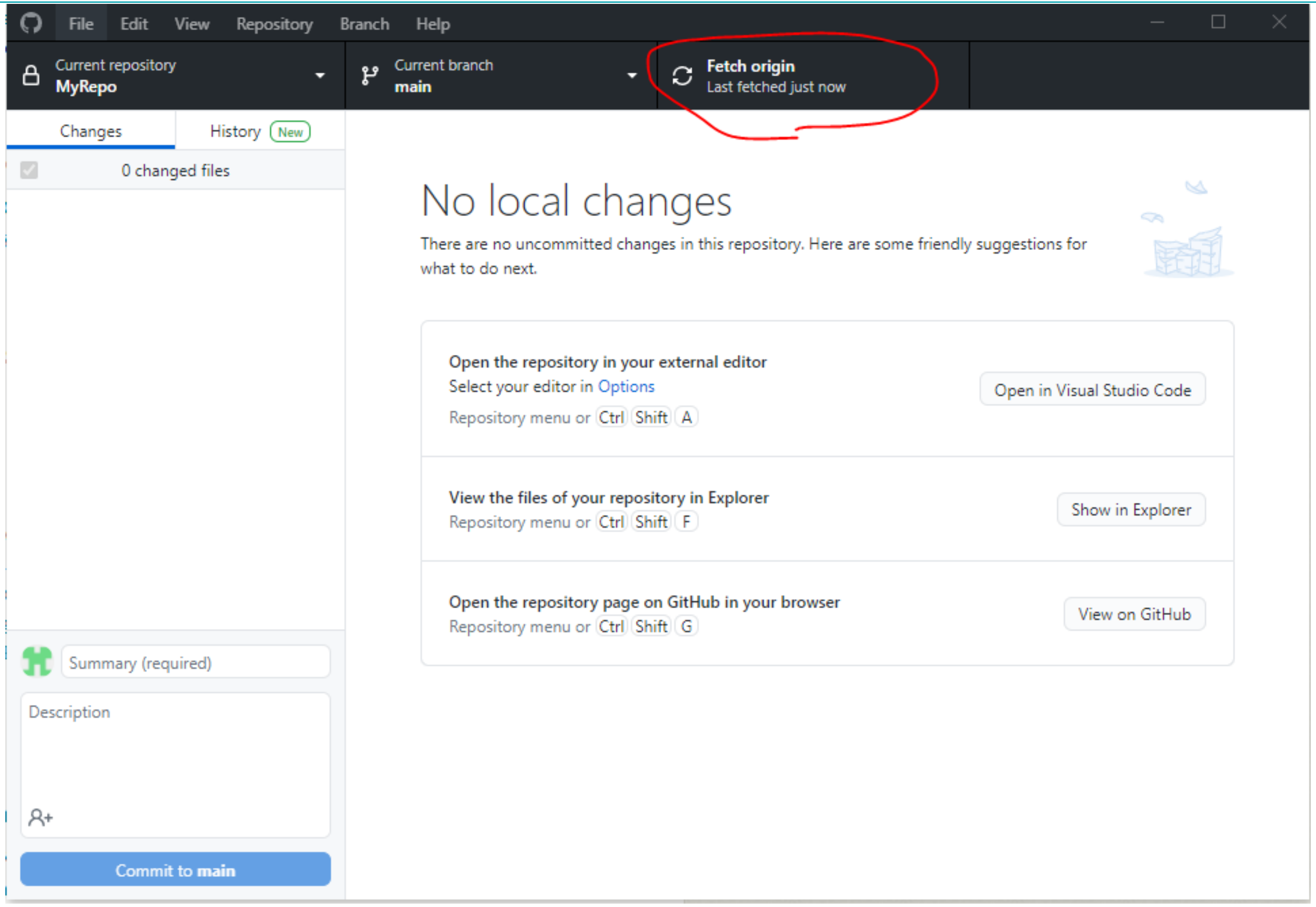
Push commits to the origin remote
You have 1 local commit waiting to be pushed to GitHub.
Always available in the toolbar when there are local commits waiting to be pushed or Ctrl P
Push origin

Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Visual Studio Code

View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer

Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub

FETCH FROM GITHUB



No local changes

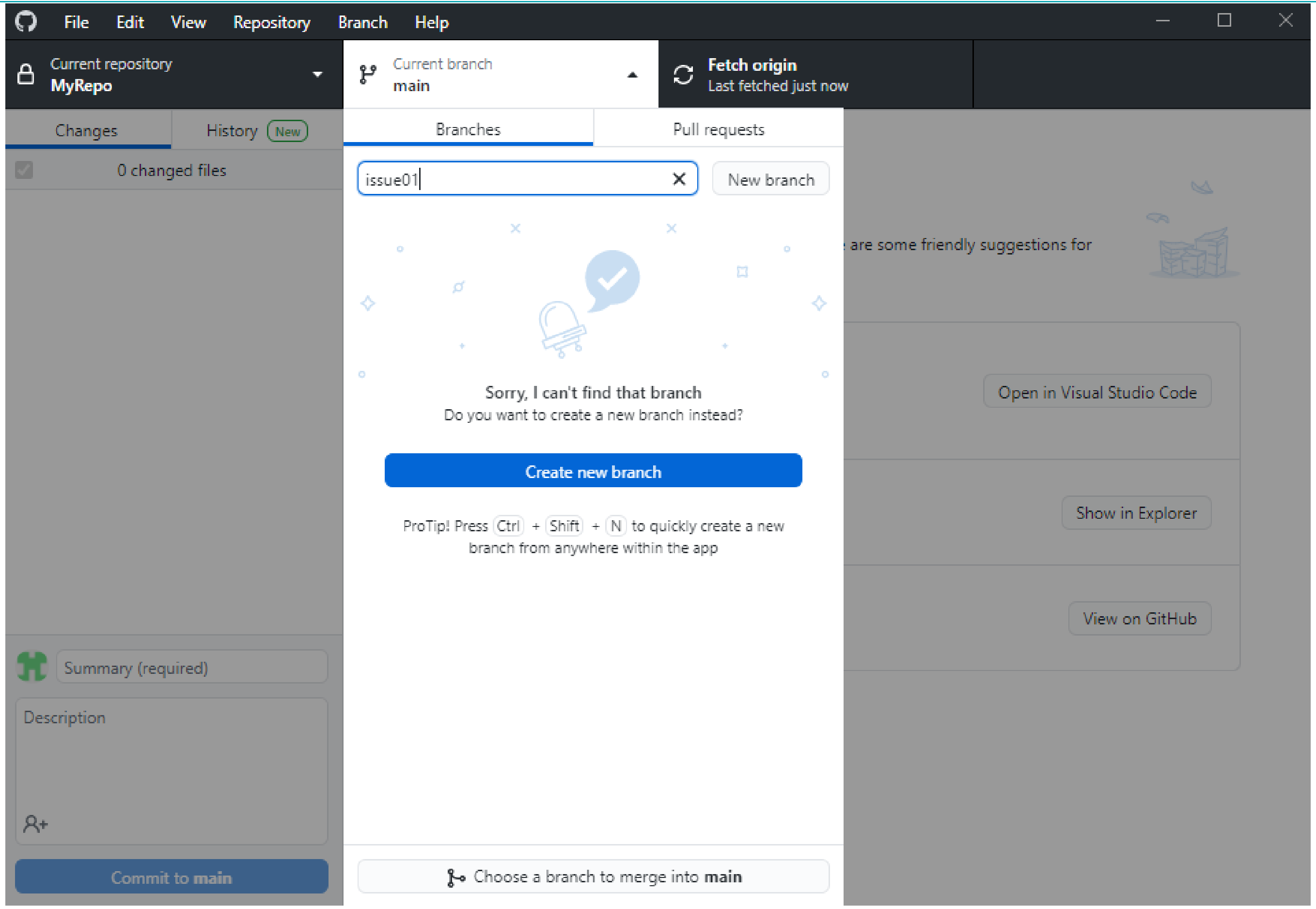
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Visual Studio Code

View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer

Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub

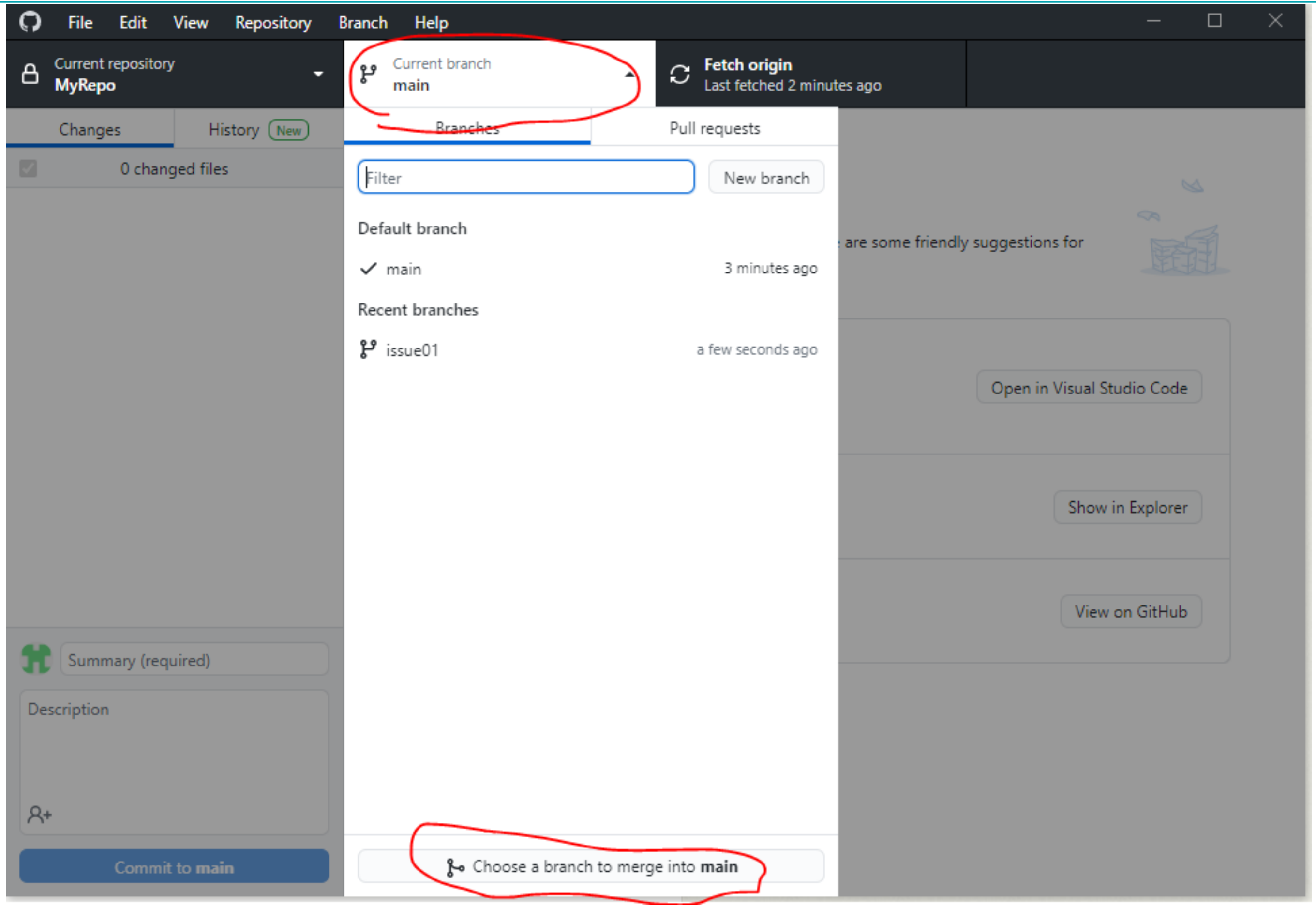
CREATE BRANCH



Sorry, I can't find that branch
Do you want to create a new branch instead?

ProTip! Press Ctrl + Shift + N to quickly create a new branch from anywhere within the app

MERGE

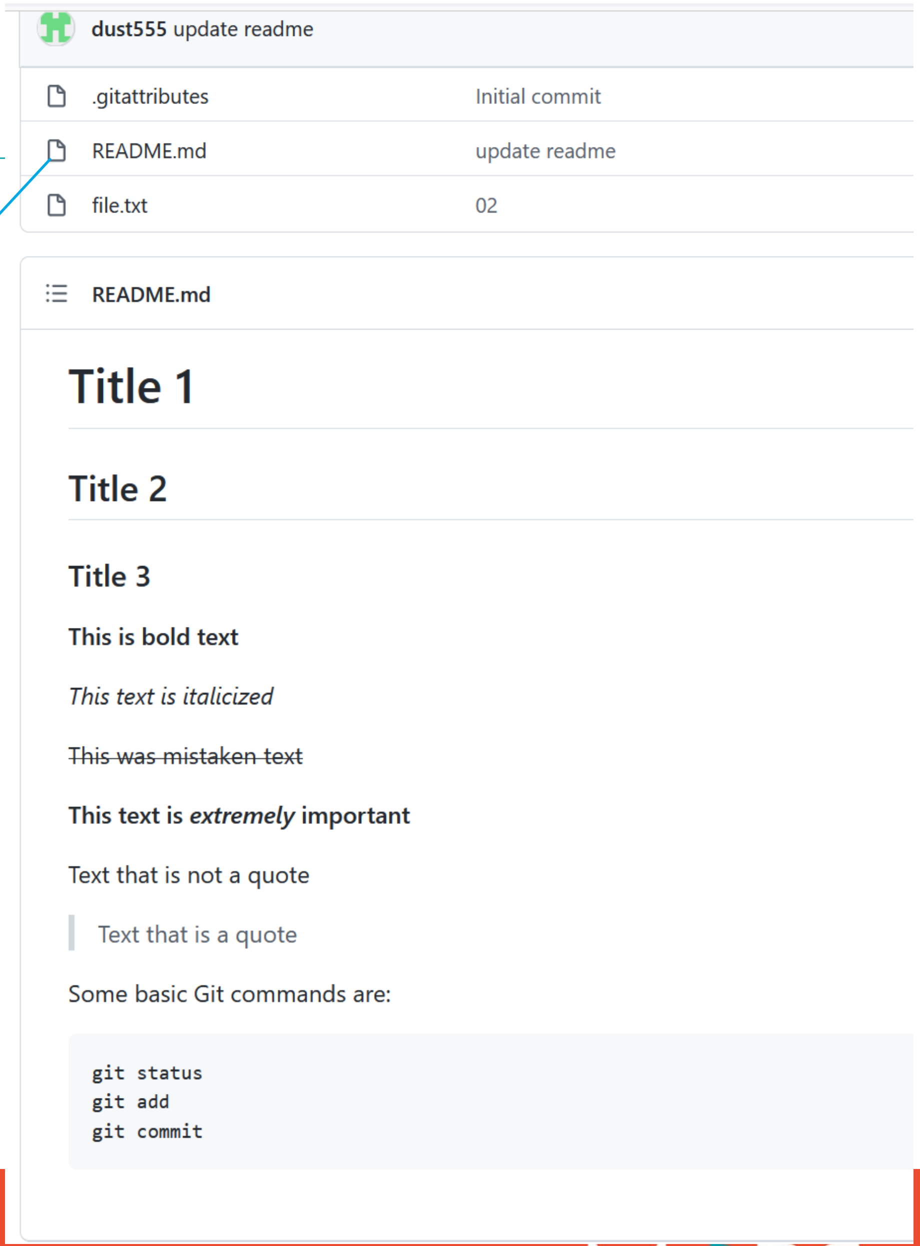


THOMAS MORE

README.MD

- Readme.md file is displayed in GitHub webpage
- Each subdir can have its own readme.md file

```
# Title 1
## Title 2
### Title 3

**This is bold text**

*This text is italicized*

~~This was mistaken text~~

**This text is _extremely_ important**

Text that is not a quote

> Text that is a quote

Some basic Git commands are:
```
git status
git add
git commit
```
```



dust555 update readme

.gitattributes	Initial commit
README.md	update readme
file.txt	02

README.md

Title 1

Title 2

Title 3

This is bold text

This text is italicized

~~This was mistaken text~~

This text is *extremely* important

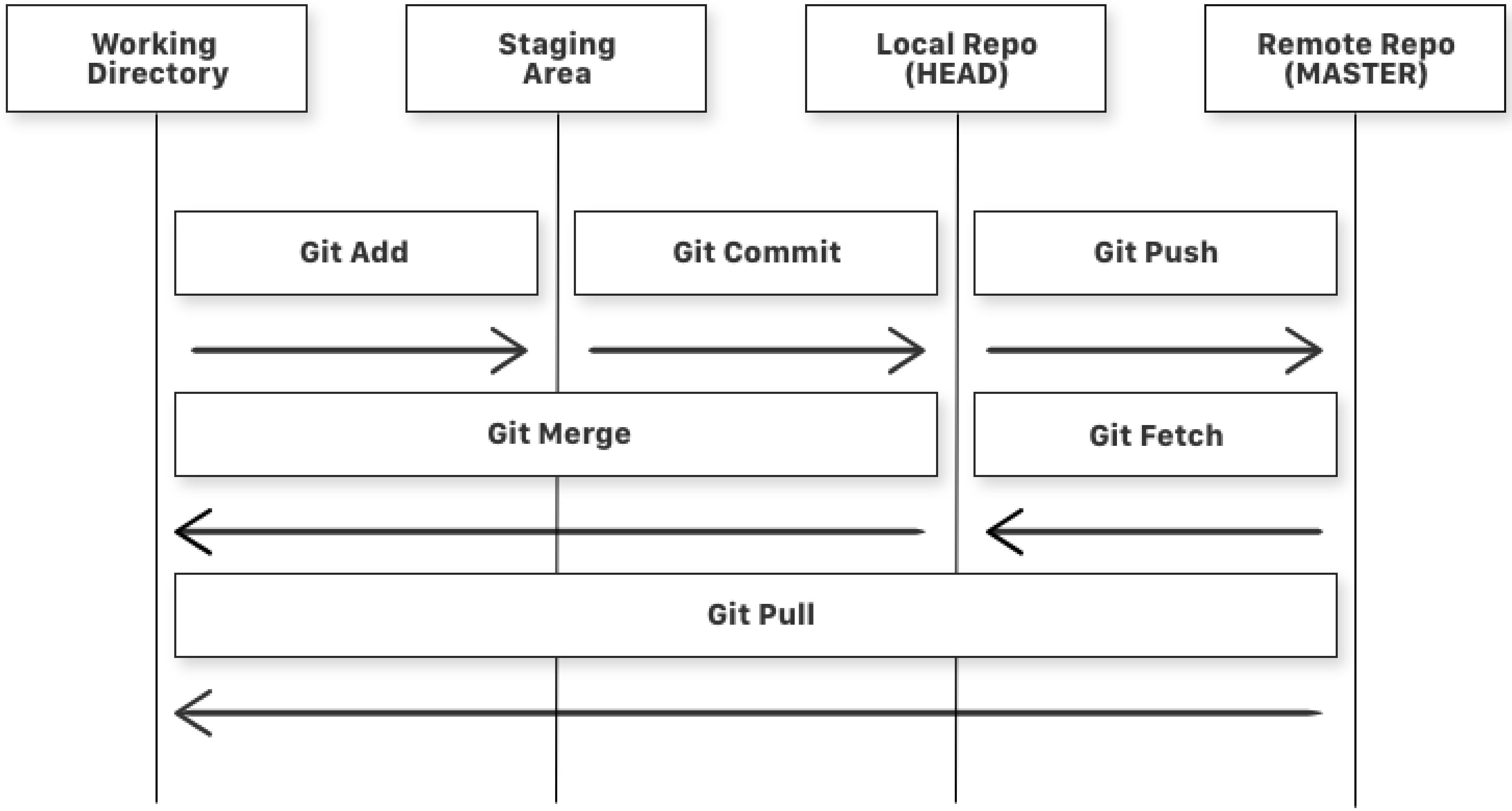
Text that is not a quote

> Text that is a quote

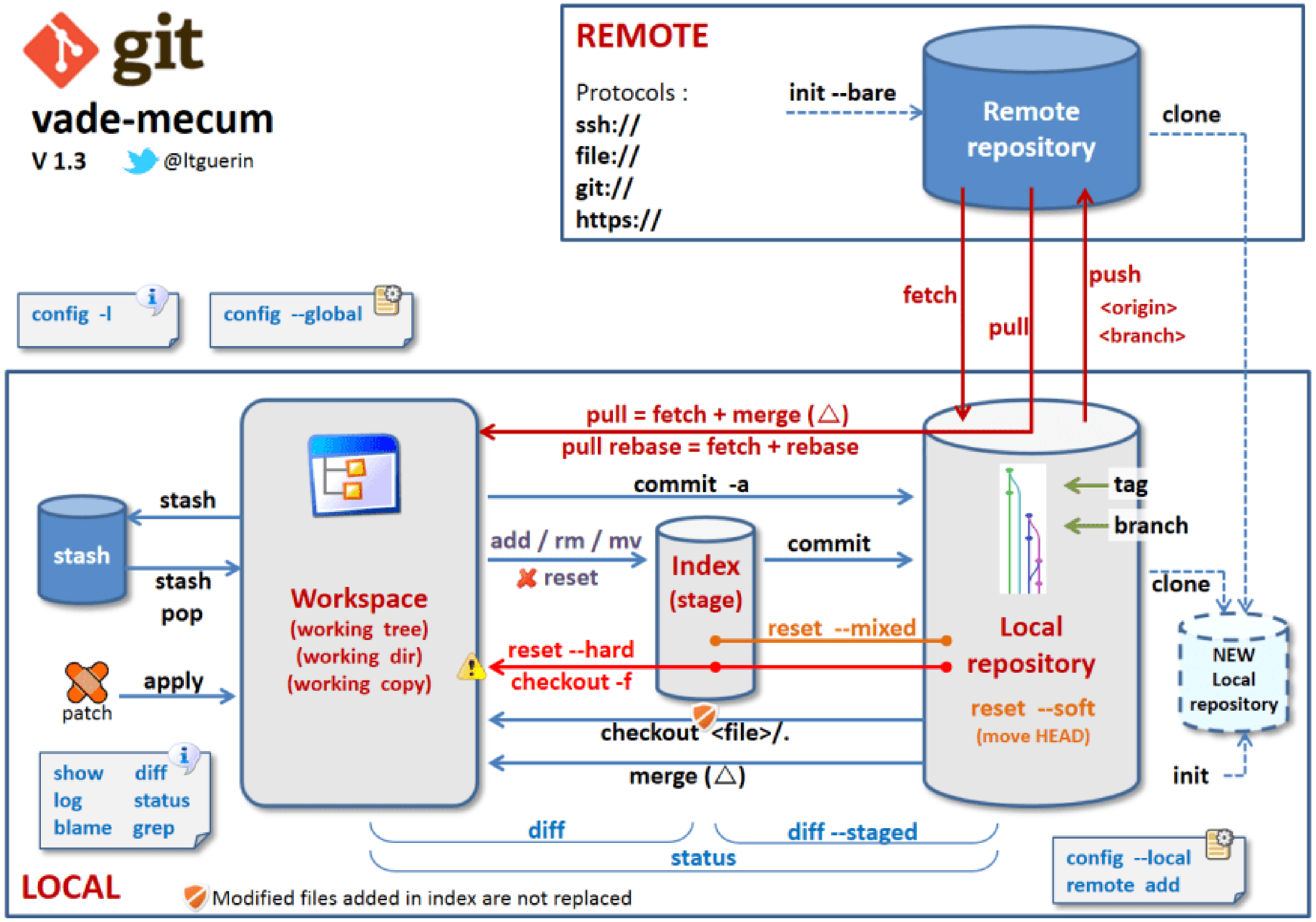
Some basic Git commands are:

```
git status
git add
git commit
```

GIT OVERZICHT

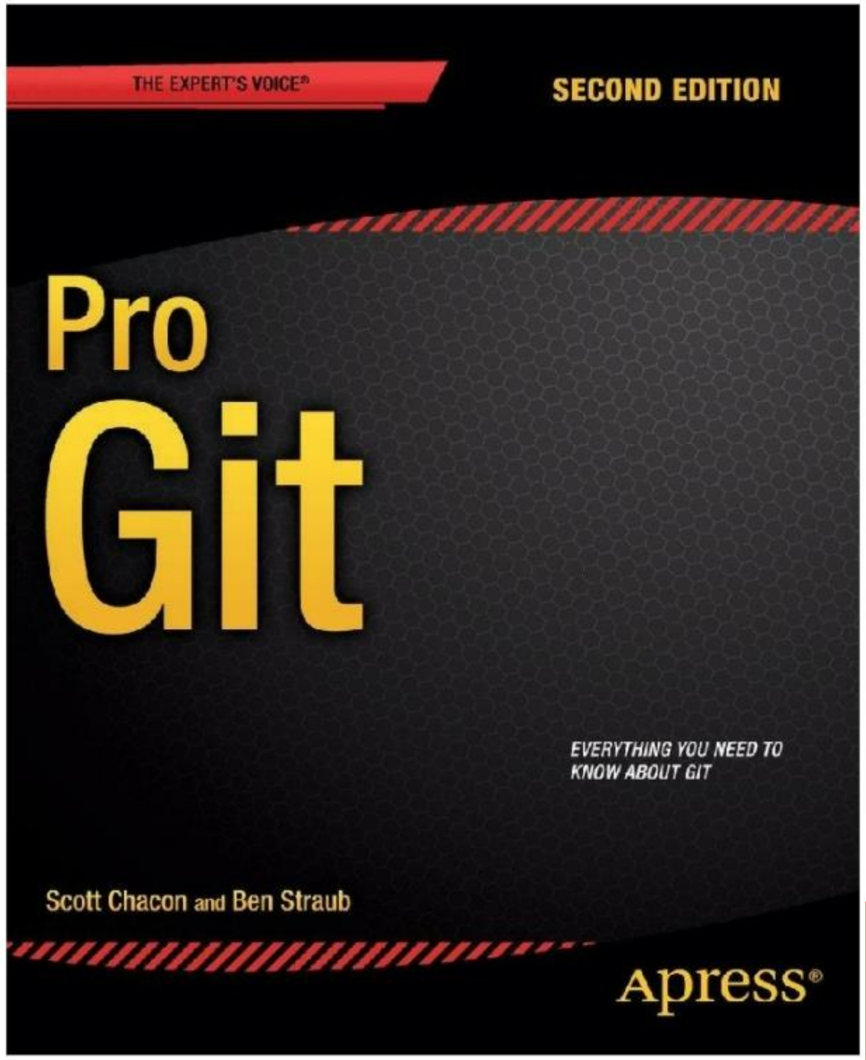


GIT OVERZICHT



SOURCE

- Slides created by Joris Dieltjens based on: https://git-scm.com/book/en/v2


THE EXPERT'S VOICE®
SECOND EDITION
Pro Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®