Find where config files are:   
# **git config --system –edit**# **git config --global –edit**Use **1+CTRL+G in vim** to show the full path.  
# git config --system **system.name** "Git Repo Server stud." **system.name** can be any thinkand can be overwritten by   
# **git config --list**

system.name=Git Repo Server stud.

user.name=nikiciu

user.email=niki.ciurlea@gmail.com

core.editor=/usr/bin/vim

system.name=Git Repo Server stud. GLOBAL

**core.excludesfile=~/.gitignore\_global**

**head -> main:** the head is currently in our main branch. Branch are used to organize project.Each branch is like an alternate reality for the project. The head always point to the current reality which is called a branch.

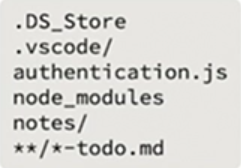
**Three environments** where you can move files:

* **Working**: files like in the last commit.
* **Staging**: temporary location before a new commit (**git add** command).
* **Commit:** a new log entry is created with a new hash.

**File states:**Before a commit a file can be in one of **two states:**

1. **tracked:** these are files that existed in the previous snapshot (commit)
   * **unmodified:**  the file hasn’t changed since the last commit
   * **modified:** the file is changed since the last commit
   * **staged:** have been moved in the staging environment
2. **untracked**: anything else (ex: a new file added since the last commit

===============================================================================  
A modified files will appear in git status as: **modified: README.md.** Changes not staged for commit. The file is not in staging environment and we can revert back the changes: **git restore README.md** " to discard changes in working directory OR we can move the file in staging using “git add filename”,   
**$ git add README.md  
$ git status: On branch master  
Changes to be committed:**use **"git restore --staged README.md " to unstage  
 modified: README.md**Now we need to move back the file from staging and restore the version from working.  
**git restore --staged README.md =>** unstaged **git restore README.md =>** the file is in its original state, status nothing to commit

**====== IGNORE FILES: ====================**In root of the directory create a .gitignore file:  
(empty folders are not tracked by default).   
This file must be added and comitted.

We can create a global ignore file pointing to a file on my hard drive. It would be useful for the same structure in every project, for ignore files like notes/, todo/, in-progress,readme.md etc. All these files and directories will be ignored if they are written in this global file. Every new project will pick-up this file automatically:  
 # **git config --global core.excludesFile [file]**If ignore files don’t work, clear (delete) all the files that are cached recursively: “**git rm -r –cached .”** followed by a new add and a new commit. Work with .gitignore file at the very beginning of when you’re setting things up!

**====================DELETING and RENAMING==============================**1. We delete a file by *right click->delete*: $ git status ;On branch master; Changes not staged for commit:

use "git add/rm <file>..." to update what will be committed

use "git restore <file>..." to discard changes in working directory

deleted: index.html

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

**$ “git restore .”** will put back the deleted file.

2. We delete a file by: # **git rm index.html** Compared to 1), this command also deleted the file but *the deletion is moved into staging.* The restore will be performed in 2 steps:   
$ git restore --staged index.html (from stage to working env)  
$ git restore index.html

RENAMING (F2 on file) it’s recording two different things: the deletion of the renamed file and the addition of a new untracked file. # “**git restore .**” will try to keep both files. Use git restore . + delete the unwanted file.

Another way to rename a file is within git: **$ git mv index.html home.html.** To undo the rename, use the same command but switch the file order: **$ git mv home.html index.html**

**REMEMBER:** git is always looking at what you are doing based on the last commit. So since we renamed the file back to what it used to be it compares everything that we done to the last commit and realizes that’s nothing that’s different=> nothing to commit.

===================DIFFERENCES==============================

# git diff =>show the difference between working directory and staging.

How to see differences between commits:

1. **git log –oneline** to shows the log in order, last change first
   1. ebd4e4e (HEAD -> master) Moved 3 .html file to docs/ folder
   2. b197576 \*\*\*\* Added .gitignore file
   3. dba02fa First commit
2. **git diff b197576**

diff --git a/columns.html b/docs/columns.html

similarity index 100%

rename from columns.html

etc……….

==============AMENDING – CHANGING HISTORY – UNDO CHANGES========================  
$ **git commit --amend** => open the last commit in editor (def editor in core.editor config).

$ **git commit -am ‘New commit message”**  => change text message directly in the terminal (no editor open).

$ **git commit --amend --no-edit** => leave the message intact as it was

$ **git log --oneline**

5ac7047 (HEAD -> master) Simplified text-uppercase in files.

b03532c Moved 3 .html file to docs/ folder

b197576 \*\*\*\* Added .gitignore file

dba02fa First commit

**$ git reset b03532c**

Unstaged changes after reset:

M docs/columns.html

M docs/index.html

M docs/media.html

This command **$ git reset b03532c** don’t delete the changes made to the files, only move the files back as in **b03532c** commit.

**$ git reset --hard b03532c** => rewind time to a checkpoint exactly as it was before

 ============= REBASING = reorganize the order each commit is made=============================  
Take the commits from one branch and apply them to another:

Move commits (events) between each other.

**$ git rebase -i --root** (-i from interactive)

The editor is open, reorganize the changes and save the file.

==============BRANCHES=======================================

Usually, we want **a copy** (-c ) of an existing branch:

$ **git switch -c NAME**

$ git checkout -b NAME => is the old version

=============== MERGE BRANCHES=============

Merge will merge the changes from one branch **into the current branch**. Switch back into the main branch and:

$ **git switch master**

$ **git merge fix-classes**

===============DELETE a BRANCH================

$ **git branch –delete NAME**

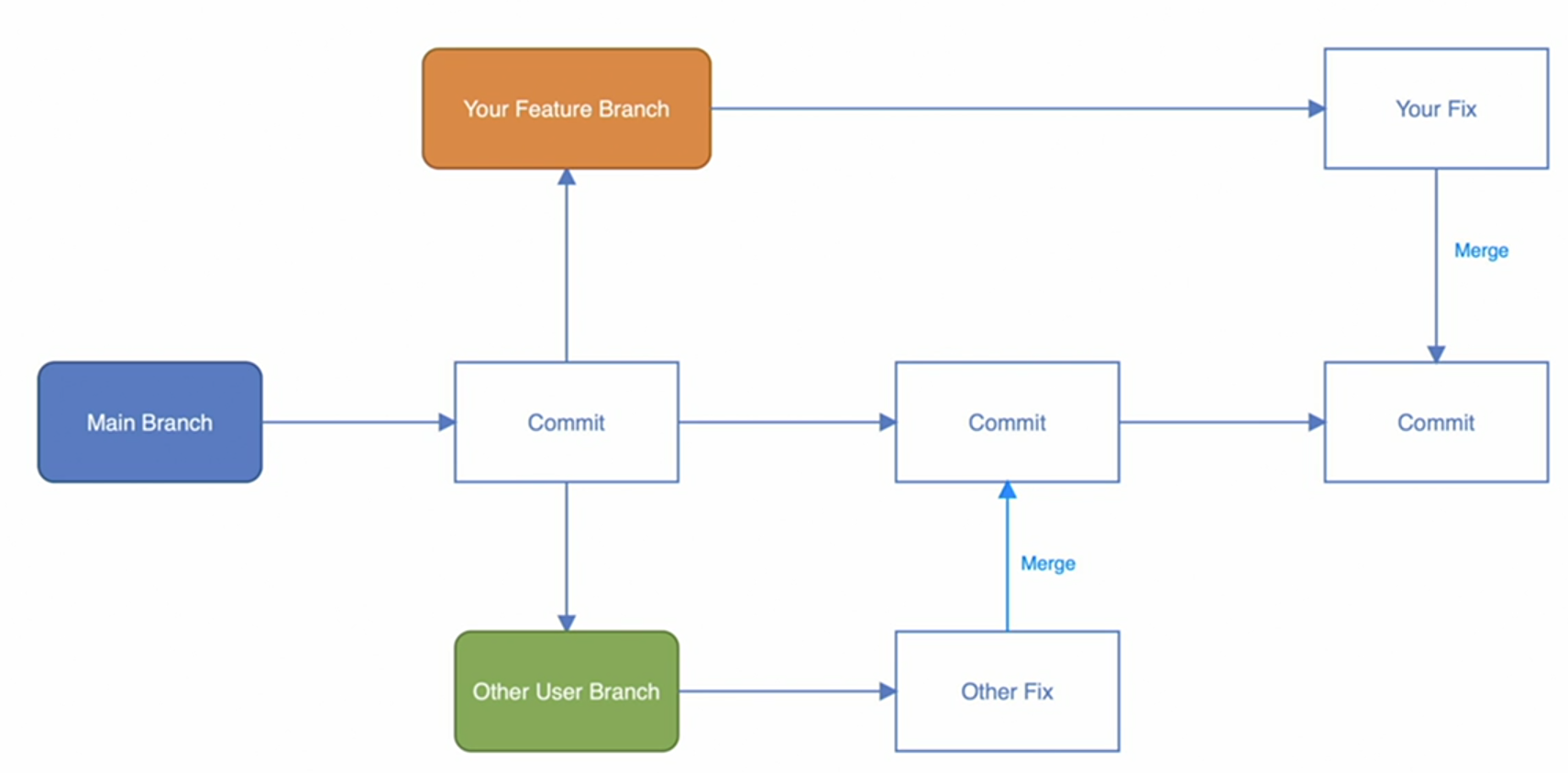
-d when branches are free of conflicts

-D ignore problems and just delete the branch



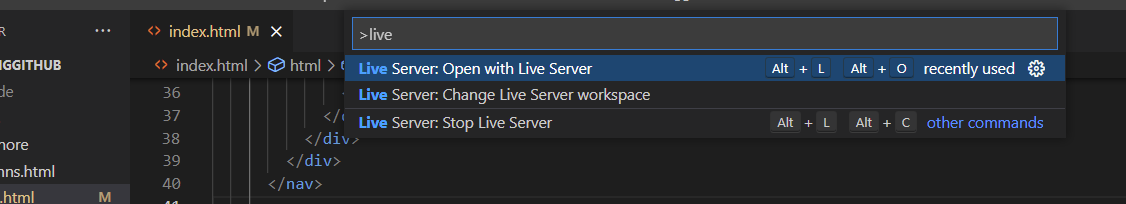
=================== MERGE CONFLICTS ============

Two users create two branches and do commit in different moments:



We can add an extension called Live Server to open the project locally.

To go te extension, type CTRL+SHIFT+p and start typing live server:



WORKFLOW:

> git init  
Initialized empty Git repository in C:/Users/cniki/Desktop/git-basic/.git/  
This will create the folder structure.

> **git status**

On branch master

No commits yet

Untracked files:

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

**initial-commit.txt**

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

PS C:\Users\cniki\Desktop\git-basic**> git add .\initial-commit.txt**

PS C:\Users\cniki\Desktop\git-basic> git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: initial-commit.txt

To add ALL dir: **git add .**

>git log

commit 7f956c176da4419967cd522b30f1ff9aab747f02 (HEAD -> master)

Author: nikiciu <niki.ciurlea@gmail.com>

Date: Mon Oct 11 16:51:05 2021 +0300

Added second .txt file

commit **858f6db1108816e992075356ae229abb8fde95c5**

Author: nikiciu <33724280+nikiciu@users.noreply.github.com>

Date: Mon Oct 11 16:43:46 2021 +0300

added first .txt file

**To revert to a previous commit:**

* **git checkout 858f6db1108816e992075356ae229abb8fde95c5**

# **Scenario:**

We create a blank project <https://github.com/nikiciu/myrepo2.git> on github.

**How do we add it on our local machine?**

$ **git remote add origin** [**https://github.com/nikiciu/myrepo2.git**](https://github.com/nikiciu/myrepo2.git)

**Verification😐**

**$ git remote show origin**

**$ git config --list | grep my**

**remote.origin.url=https://github.com/nikiciu/myrepo2.git**

\* remote origin

Fetch URL: https://github.com/nikiciu/myrepo2.git

Push URL: <https://github.com/nikiciu/myrepo2.git>

$ **git config --list | grep my**

remote.origin.url=https://github.com/nikiciu/myrepo2.git

**$ git config --edit** ==>**1+Ctrl+G** reveal the file path.

[core]

repositoryformatversion = 0

filemode = false

bare = false

logallrefupdates = true

symlinks = false

ignorecase = true

[remote "origin"]

**url = https://github.com/nikiciu/myrepo2.git**

fetch = +refs/heads/\*:refs/remotes/origin/\*

**Push the project to github:**

$ **git push -u origin master 🡺** (-u, --set-**u**pstream set upstream for git pull/status). Use -u only one time, when we first push, after then only git push will be enough.

Make changes. Commit changes. Repeat: **$ git push -v -u origin master**

**Now, the project is on github. Other users who want to work on it must CLONE it! Make changes,commit…**

**$ git clone** [**https://github.com/nikiciu/myrepo2.git**](https://github.com/nikiciu/myrepo2.git)

**fork**: copy others project to your github account:

Before push changes, do ALWAYS a pull:

# **git pull origin master**

* add and commit commands can be done into **one single step**:
* works only for already **tracked** files that can be found with:

**$** **git ls-files => all files being tracked in the current repository (new files don’t appear here)**

**$ git commit -am “Adding more ipsum text” => “a” from add**

We add a file to the staging area (git add) and then we modified it again What git status report?

The same file two times: one unstaged and one staged:

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hipster.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: hipster.txt

**Recursively add files to git repo:**

$ mkdir -p level1/level2/level3/level5

Create some file in each directory: level1-file.txt, level2-file.txt,level3-file.txt

Git status won’t show all files recursively, only the first level directory!!:

To add them all, use add with . option: “**git add .**”

$ git status

On branch master

Your branch is ahead of 'origin/master' by 3 commits.

(use "git push" to publish your local commits)

Untracked files:

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

**level1**/

$ git add .

$ git status

On branch master

Your branch is ahead of 'origin/master' by 3 commits.

(use "git push" to publish your local commits)

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: level1/level1-file.txt

new file: level1/level2/level2-file.txt

new file: level1/level2/level3/level3-file.txt

**Revert changes**

Starting from a clean working directory => nothing to commit, working tree clean

* Change a file.
* Add it to the staging area: git add level1-file.txt. **The file is in staging area** (not commited) and I decide I don’t want these changes and I want to revert them all (unstage).
* $ git restore --staged (file is now unstaged but unchanged)
* $ git restore level1-file.txt => DONE, file in the initial state.

**COMPARE/MERGE Tool**

Visual diff tool: P4Merge (visual compare and visual merge tool)  
<https://www.perforce.com/downloads/visual-merge-tool>

$ git config --global --list | grep tool

diff.tool=p4merge

difftool.p4merge.path=C:/Program Files/Perforce/p4merge.exe

difftool.prompt=false

merge.tool=p4merge

mergetool.prompt=false

mergetool.p4merge.path=C:/Program Files/Perforce/p4merge.exe

Fork de aici: <https://github.com/awesomejt/starter-web> in contul meu de github, apoi clonare locala.

$ git clone <https://github.com/nikiciu/starter-web.git>

$ git remote show origin 🡺 Este OK?

$ **git diff** 🡺Compare the differences between what’s in my local working directory that has been recently modified but not yet staged versus what’s currently staged (compare red and green files):

$ git status

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: README.md

Changes not staged for commit:

modified: README.md

$ git diff (diff. between working dir. and staged)

$ git diff

**diff --git a/README.md b/README.md**

**index 1313a14..73f766c 100644**

**--- a/README.md**

**+++ b/README.md**

@@ -12,4 +12,6 @@ As stated above, the main purpose is to provide simple examples for it training

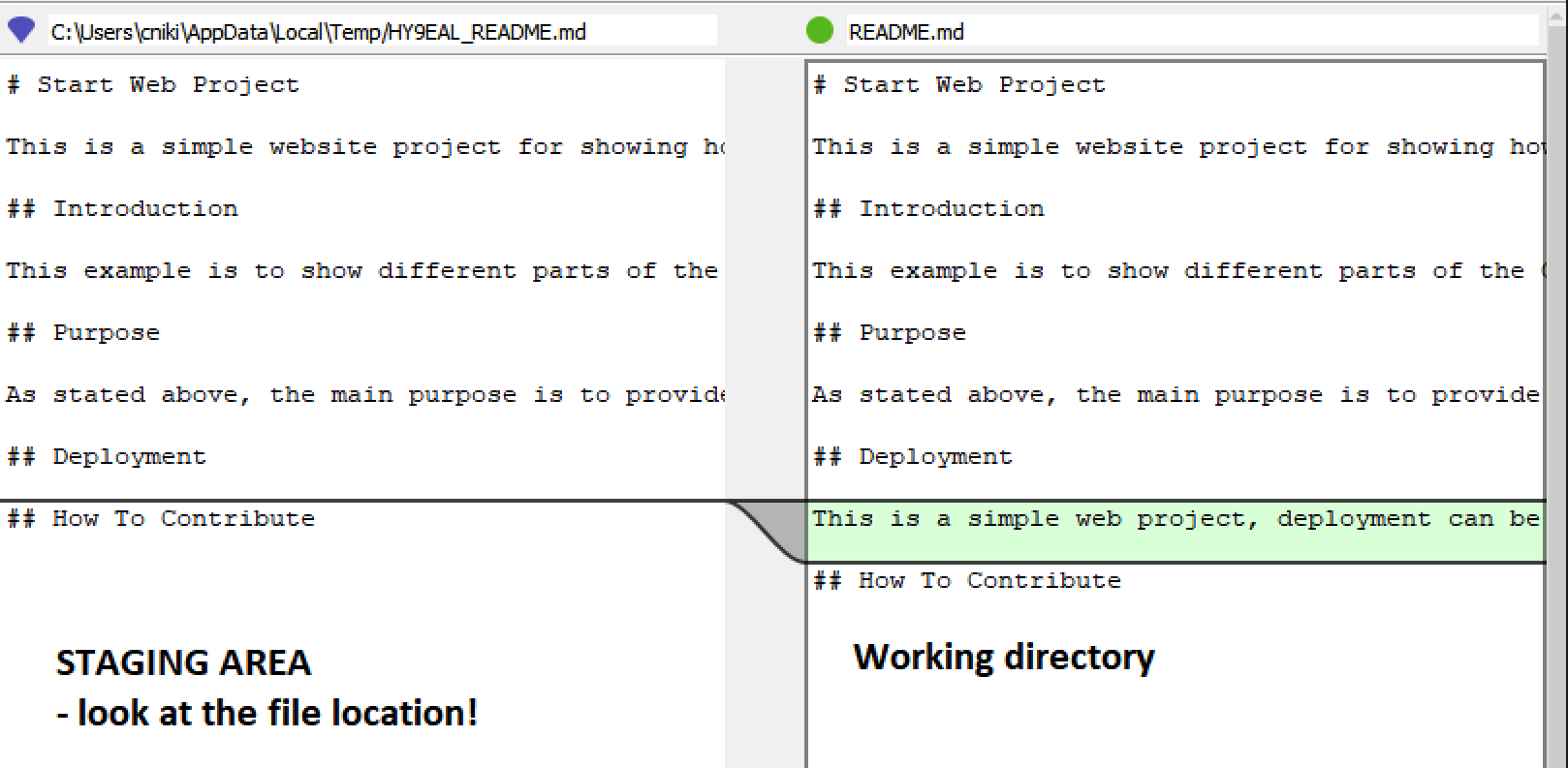
## Deployment

+This is a simple web project, deployment can be on any web server or even local file system.

+

## How To Contribute

Now we use the visual difftool. The same files are compared:



(the temp location is used by the difftool temporarly.)

**Differences between the working directory and the last known commit: git diff HEAD**

**If we use the pointer reference HEAD as a parameter, git will compare differences between the working directory and last commit on this branch. Last commit change is located in .git repository!**

**$ git diff HEAD**

**diff --git a/README.md b/README.md**

**index 6c18120..73f766c 100644**

**--- a/README.md**

**+++ b/README.md**

@@ -8,6 +8,10 @@ This example is to show different parts of the Git repository and various comman

## Purpose

+As stated above, the main purpose is to provide simple examples for it training demos.

+

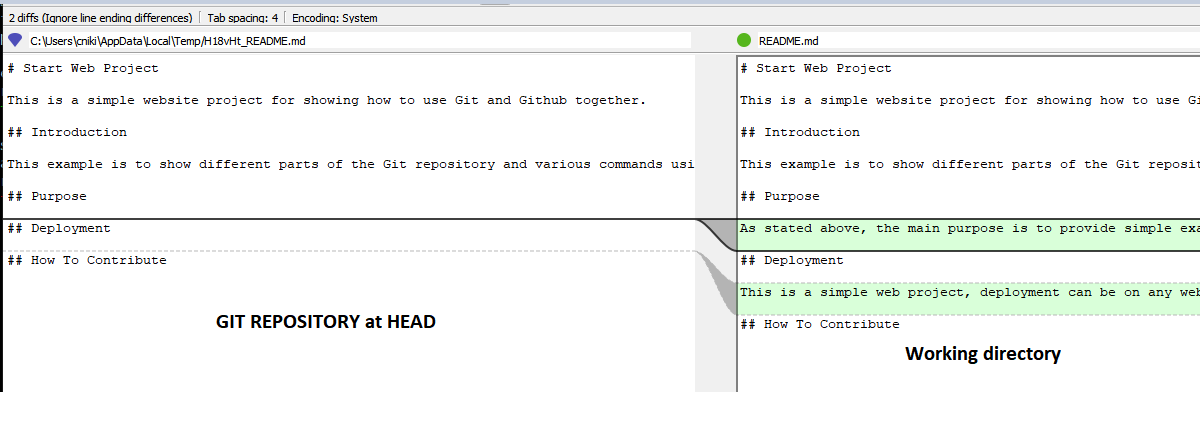
## Deployment

+This is a simple web project, deployment can be on any web server or even local file system.

+

## How To Contribute

Now, the visual diff tool: **$ git difftool HEAD**



**Differences between what’s been staged and the last commit: git diff --staged HEAD**

**$ git diff --staged HEAD**

**diff --git a/README.md b/README.md**

**index 6c18120..1313a14 100644**

**--- a/README.md**

**+++ b/README.md**

@@ -8,6 +8,8 @@ This example is to show different parts of the Git repository and various comman

## Purpose

+As stated above, the main purpose is to provide simple examples for it training demos.

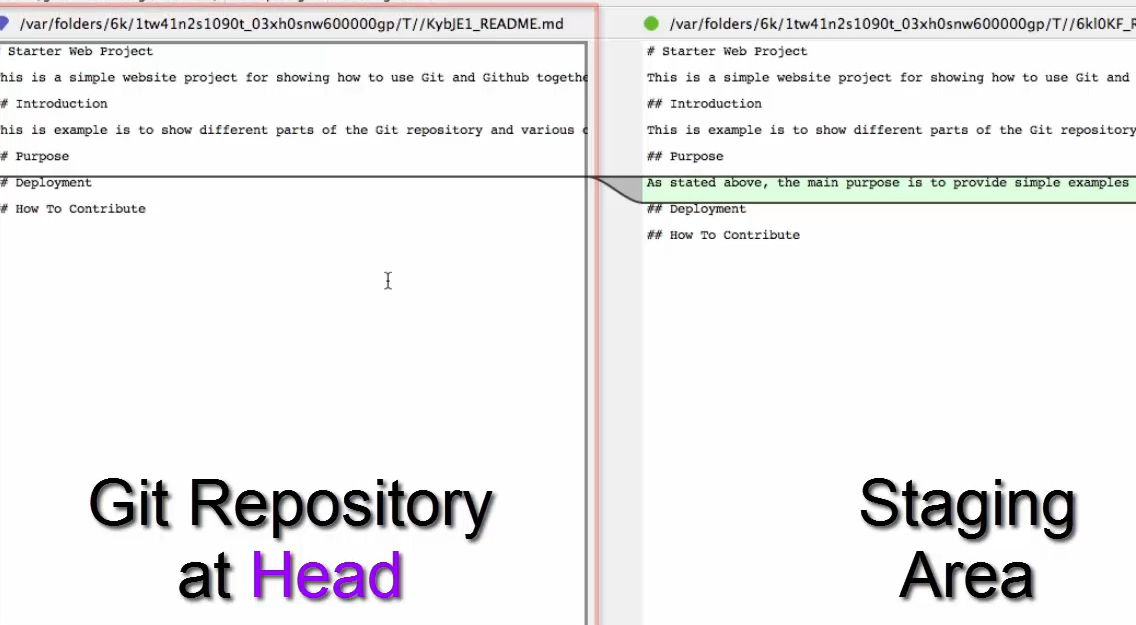
+

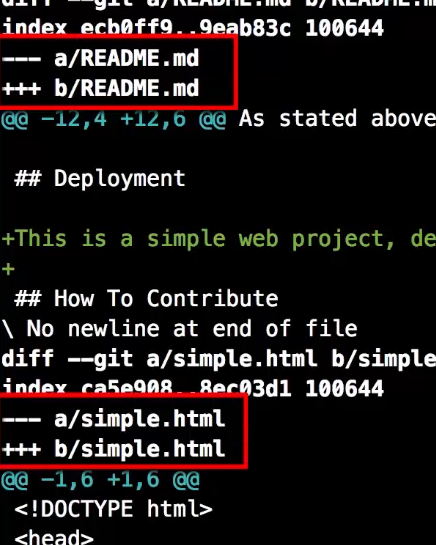
## Deployment

## How To Contribute

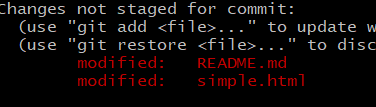
**The HEAD means the last commit in the current branch!!!**

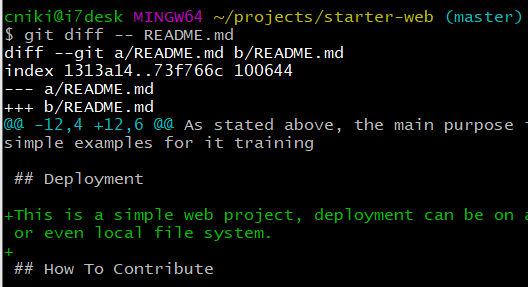
**$ git diff --staged HEAD**



**Sometimes the diff involves multiple files and we want to limit the search just one file.**

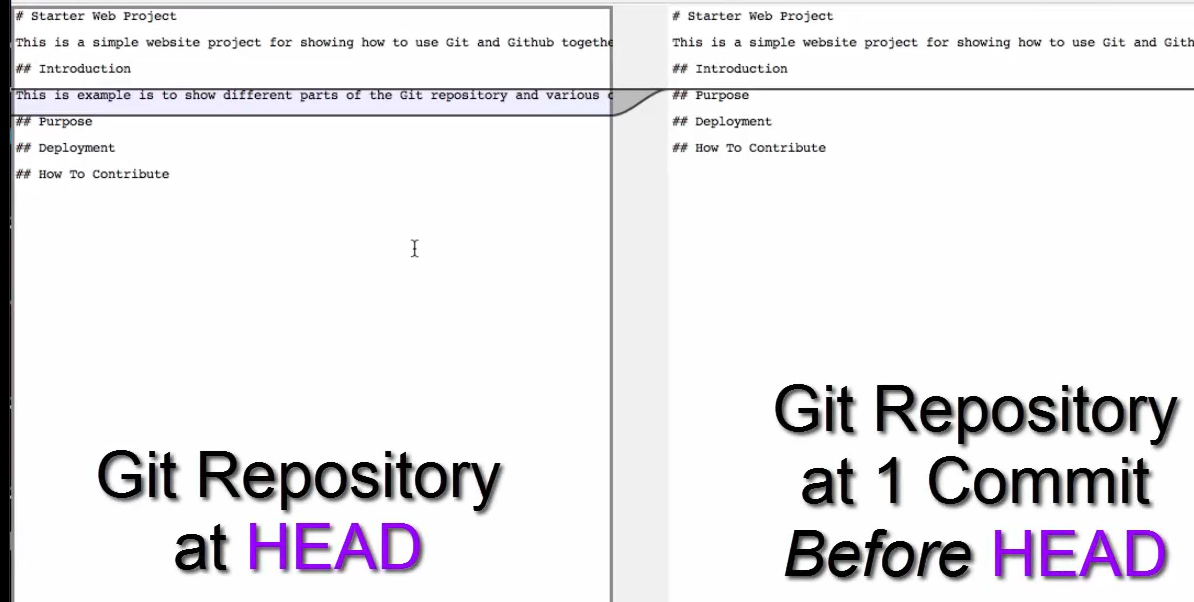
**Limit the diff to just one file - the README.md file:**

 **$ diff -- README.md**



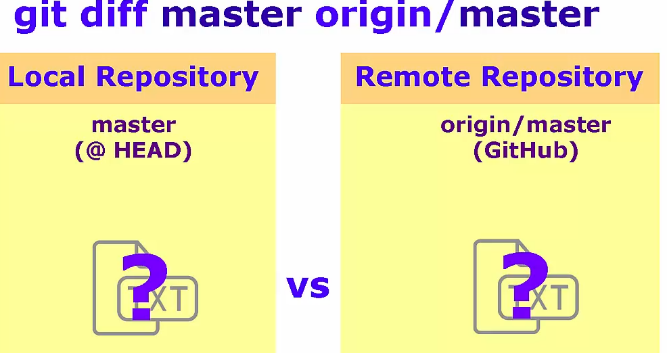
|  |  |
| --- | --- |
| git diff | Differences between what’s in my local working directory that has been recently modified but not yet staged (red) versus what’s currently staged (green). |
| git diff -- README.md | Limit differences to just one file (working area - staging area) |
| git diff HEAD | Differences between the working directory and the last known commit. |
| git diff --staged HEAD | Differences between what’s been staged and the last commit. |
| git diff commit1 commit2 | Diff. between two commits. |
| git diff commit1 HEAD | Diff. between commit1 and pointer HEAD (which points to the last commit) |
| $ git difftool HEAD HEAD^ | Compare commited files HEAD and HEAD-1 |
| $ git diff -- index.html | If there are many files, limit the diff to only one file (index.html) |

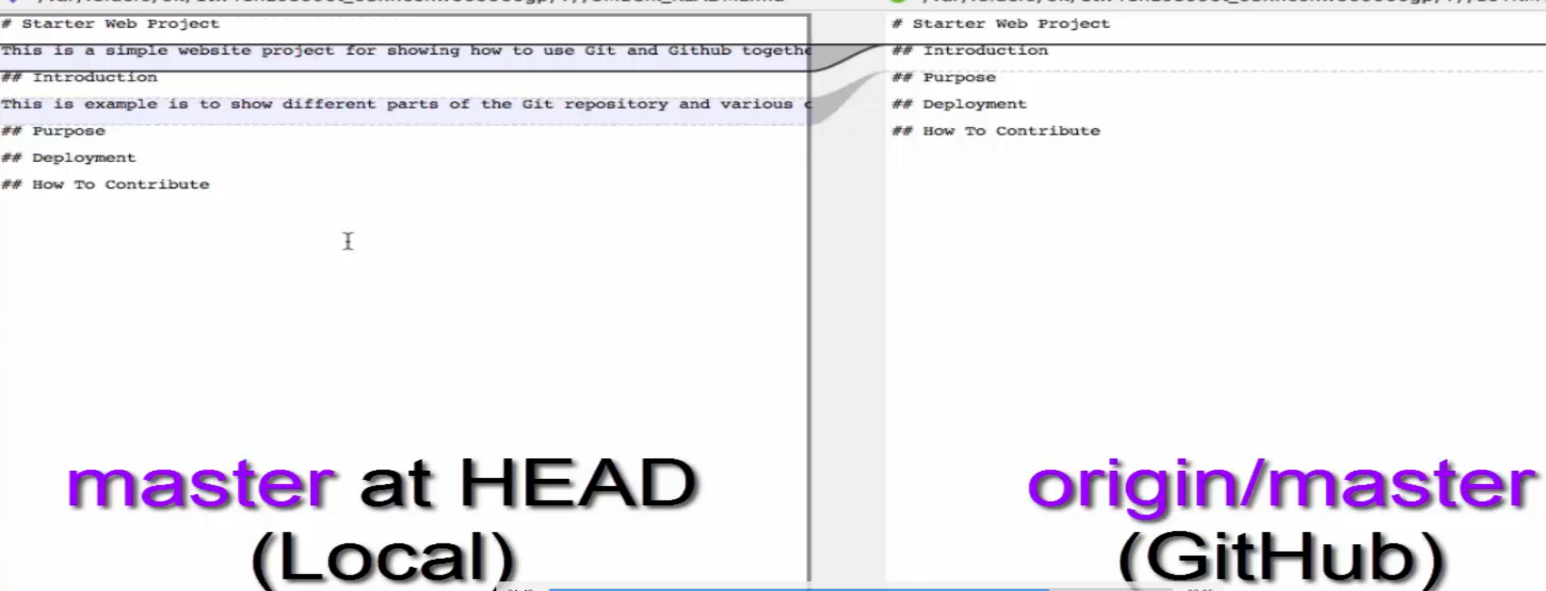
**# git diff HEAD HEAD^**



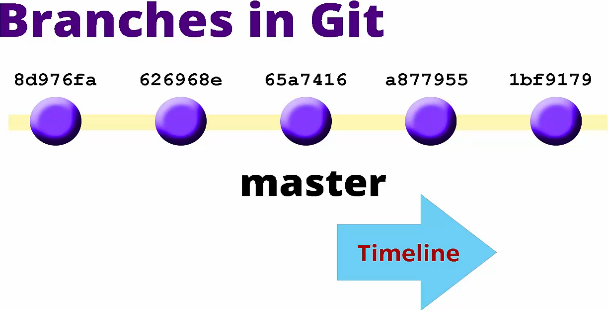
**Differences between two branches**

List branches:  
Local master branch working dir. (local repo) and remote master branch:

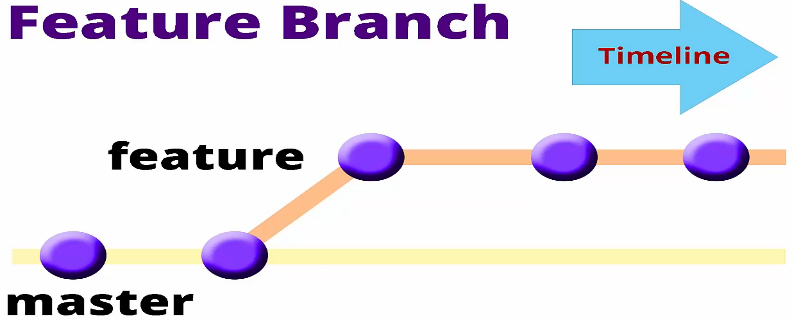
**$ git diff master origin/master**



**BRANCHES AND GIT**



So far all our changes was done on master => NOT best practice.

We should separate the changes into branches.

**$ git** **branch**  = list local branch

**$ git** **branch -a**  = list local AND remote branch

**$ git** **branch mynewbranch**  = create a new branch

**$ git** **checkout mynewbranch** = switch to mynewbranch

**$ git branch -m mynewbranch newbranch =** Rename a branch by **m**oving it

**$ git branch -d newbranch = d**elete a branch

**$ git checkout -b title-change** = create and switch to a new branch

***FLOW***: **Create a new branch. Make the changes, commit. Swithc back to master. Check the diferences. Merge. Delete unused branch.**

**$ git switch master**

**$ git diff master title-change**

**$ git merge title-change**

Updating 0b0f6a3..ff87f31

Fast-forward 🡺 no changes being made on the target branch

simple.html | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)🡺 git see a line change as a line deletion and a line insertion at the same place

**Fast-forward** merge => git placed all the commits on the master branch as if we never branched away. Only possible when no changes occured on the target branch.

**$ git log --oneline --graph –decorate  
$ git branch --delete title-change**

**If we don’t have any changes on the target branch, the merge is done fast-forward.**

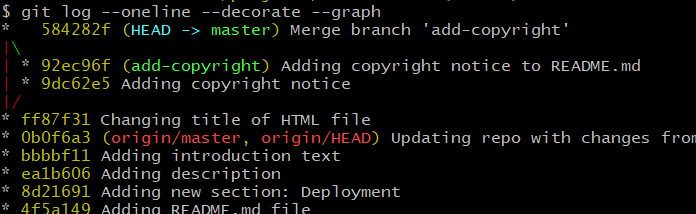
On a clean master branch, create a new branch and switch into it:

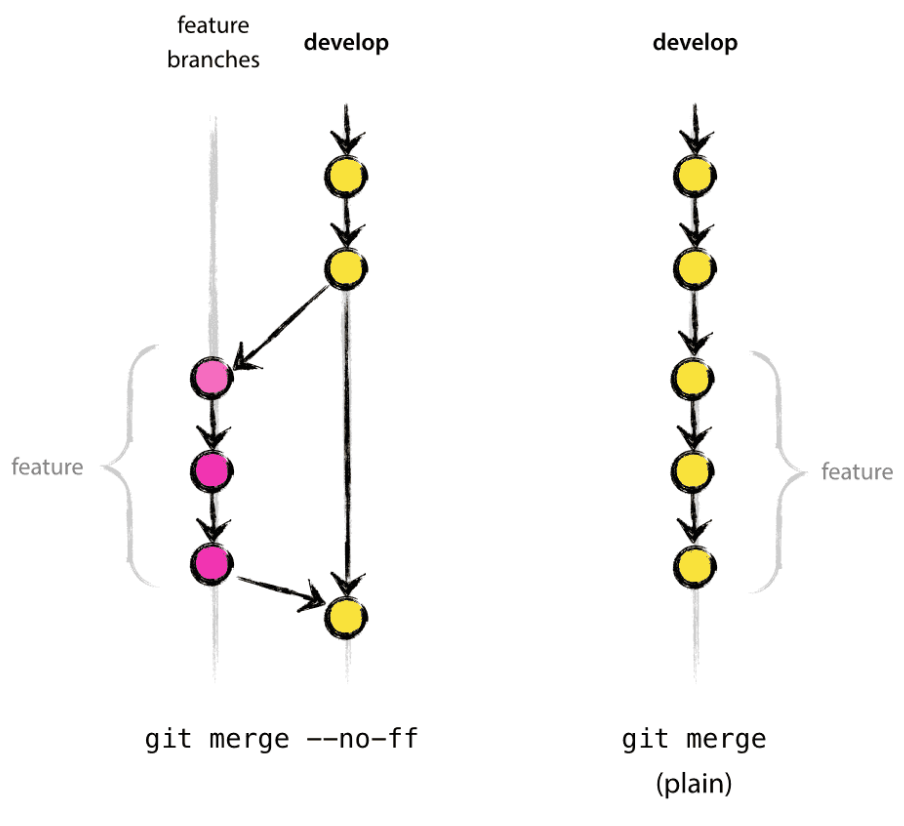
**$ git checkout -b add-copyright**

Change the file simple.html file, commit.  
Change the file README.md file, commit.  
Integrate changes to master branch: switch to master, **merge without Fast Forward (--no-ff):**

**$ git merge add-copyright --no-ff**

Fast Forward will reveal the fact that we create a new branch,  
(branched-off) and merged back the changes.  
This will result with a merge commit.





$ git log --oneline --decorate --graph

\* 584282f (HEAD -> master) Merge branch 'add-copyright'

|\

| \* 92ec96f (**add-copyright**) Adding copyright notice to README.md

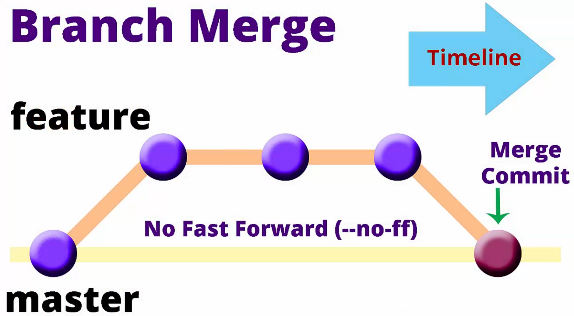
| \* 9dc62e5 Adding copyright notice

|/

After deleting the future branch:

$ git log --oneline --decorate --graph

\* 584282f (HEAD -> master) Merge branch 'add-copyright'

|\

| \* 92ec96f Adding copyright notice to README.md

| \* 9dc62e5 Adding copyright notice

|/

\* ff87f31 Changing title of HTML file

**AUTOMATIC MERGES**