Git Centralized Workflow

The following document describes git centralized workflow. Git centralized workflow is a simple workflow that is suitable for small teams. *It is by no means the only workflow. If you and your team are already comfortable working with git, then by all means, feel free to use a workflow that you are more comfortable to work with.*

If you are a beginner to git, this document is meant for you

Follow the instructions in the document `Project_Git_Repo_Creation.pdf` in the documents section of the workspace to clone and create a copy of the repository for your project

Git is a version control system that allows teams to have a centralized server side copy of their project, while at the same time allowing each member of the team a local copy of the same project on their computer. Git allows you to make changes and work on your code on your local computer without any impact to the remote project. Only after you have written and tested your code as working, should you add your changes to the remote project. In the following section we will define some basic terms in git.

Basics

Getting Started

- **Project** - A project is your software project. You will have a local personal copy of the project on your PC and git will have a remote copy of the project on a server. In our case it will be in https://gitlab.metropolia.fi
- **Repository(Repo)** - A directory in your project called `.git/` that keeps tracks of all

of the changes that you have made to a project. A repository would be created using the `git init` command **(This should not be used if you are cloning a git project as it will already have a `.git/` directory)**.

- **Branch** - A branch can be considered as a single line of development. These instructions will only consider a single main branch for all developers.
- **HEAD** - Reference to the current snapshot or state. The HEAD can point to previous commits on a branch.
- **clone** - Typically done one time to copy a git project from a remote source. Cloning is usually done once to get a working copy of a project. Once the project is cloned all modifications, additions, and removals are managed on your local machine.

Managing Changes on you local computer

- **status** - `git status` command gives you an output of anything that has changed in your local project's directory or staging area

Use git status often, it is very helpful!

```
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:   test.py

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

- **add** - Running the `git add` informs git of which changed files and/or directories you wish to be staged (included in the changes that will be reflected in the remote project eventually) `git add .` will add all files and directories that have been indicated in `git status` If you wish to add particular files you can use: `git add <file/directory name>`

```
git add .

#after adding all files take a look at git status

git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   test.py
```

- diff - The `git diff` command allows you to see changes between different, files, commits etc. If you run the command `git diff --cached` after adding files to the staging area, you will see an output that highlights all of the changes that have been made as well as where they were made. A `-` sign indicates a removal, and a `+` indicates an insertion. As can be seen from the following output the only change that has occurred is that the Pin of my LED has been changed from "LED" to 20. If files are already staged (you have added the with `git add`, then you will need to include the `--cached` flag) If you have not staged the files for commit, then you can just run `git diff` and you will get an output similar to below

```
diff --git a/test.py b/test.py
index 7c53477..4371860 100644
--- a/test.py
+++ b/test.py
@@ -14,7 +14,7 @@ print('I am test.py')
 if rb.empty():
     print('Fifo is empty')

-led = Pin("LED", Pin.OUT)
+led = Pin(20, Pin.OUT)

 while True:
     led.toggle()
```

- commit - Is taking a snapshot of your current project. It will only take a snapshot of files and directories that have been added using the `git add` command. Once files are added you can run `git commit -m "" Your commit messages should be descriptive, and commits should be incremental. It is better to commit small changes often, than it is to make giant commits. Git will help you keep track of all of your changes

```
└─ git commit -m "Changed LED from the onboard PicoW LED to pin 20"
[main 9d1a38d] Changed LED from the onboard PicoW LED to pin 20
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- **log** - `git log` shows you a list of your commit history. At anytime you can go back to a previous commit by making a note of the hashed commit ID or you can check what is different with two commits by using `git diff <commit ID 1> <commit ID 2>`. By default the IDs are long so you can invoke eht `--abbrev-commit` or `--oneline` flag to get shorter hashes

Example of `git log` output

```
commit 9d1a38d2a927e3329fe8be9dc33c4b439a9ad2d0 (HEAD -> main) #NOTE THE
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Sat Nov 11 14:30:35 2023 +0200

    Changed LED from the onboard PicoW LED to pin 20 # NOTE THE COMMIT M

commit e761c45b6d49722ef87d3faba48d6f84b0d11413 (origin/main)
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Thu Nov 9 11:42:57 2023 +0200

    added comment to test.py

...
```

Example of `git log --oneline` output

```
ent to test.py
bb439a1 (old_origin/main, old_origin/HEAD) Pulled library update
3d3fec1 Updated lib
fa87a8a Added mqtt
cdf833d Changed wmic to taskkill and added title for local webserver
0120b45 Moved web server start earlier and added a delay before running
f7d4882 Moved web server start earlier and added a delay before running
bd47a39 Changed order of commands and display detected device name
2fa0d37 Changed order of commands, echo off, and display detected device
45445e9 Added more instructions
47e3a55 Added more instructions
1fd2e56 Added more instructions
c64d74d Added more instructions
7d41535 updated README
bf0763b Changed mpremote invoke style
c21adf5 Prevent MinGW path expansion during Pico update
06f765e Changed output wording
302cefd removed obsolete scripts
973c7a1 changed permissions on install.sh
ec14586 Added notification what pyhton executeable is used
bc66dd9 improved OSX python detection
6613789 Added python3.12 detection to OSX
4021478 Fixed missing file
7ab0475 renamed main.py to prevent automatic start at boot
9929e12 Changed shell script to work on git bash also
c9b8b97 Fetched updates
af9e1c1 Improved windows install script
cefa9eb Get port in OSX or Linux
0e7cf4b Added OSX/Linux install script
fef0e0d Added pico install script for windows
8ce238e Added pico-lib
1be5091 Initial commit
```

Example of `git log --abbrev-commit` output

```
commit 9d1a38d (HEAD -> main) # NOTE THE SHORT HASH
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Sat Nov 11 14:30:35 2023 +0200

    Changed LED from the onboard PicoW LED to pin 20 # NOTE COMMIT MESSS

commit e761c45 (origin/main)
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Thu Nov 9 11:42:57 2023 +0200

    added comment to test.py

commit bb439a1 (old_origin/main, old_origin/HEAD)
Author: Keijo Länsikunnas <keijo.lansikunnas@metropolia.fi>
Date:   Mon Oct 30 23:59:54 2023 +0200

    Pulled library update
```

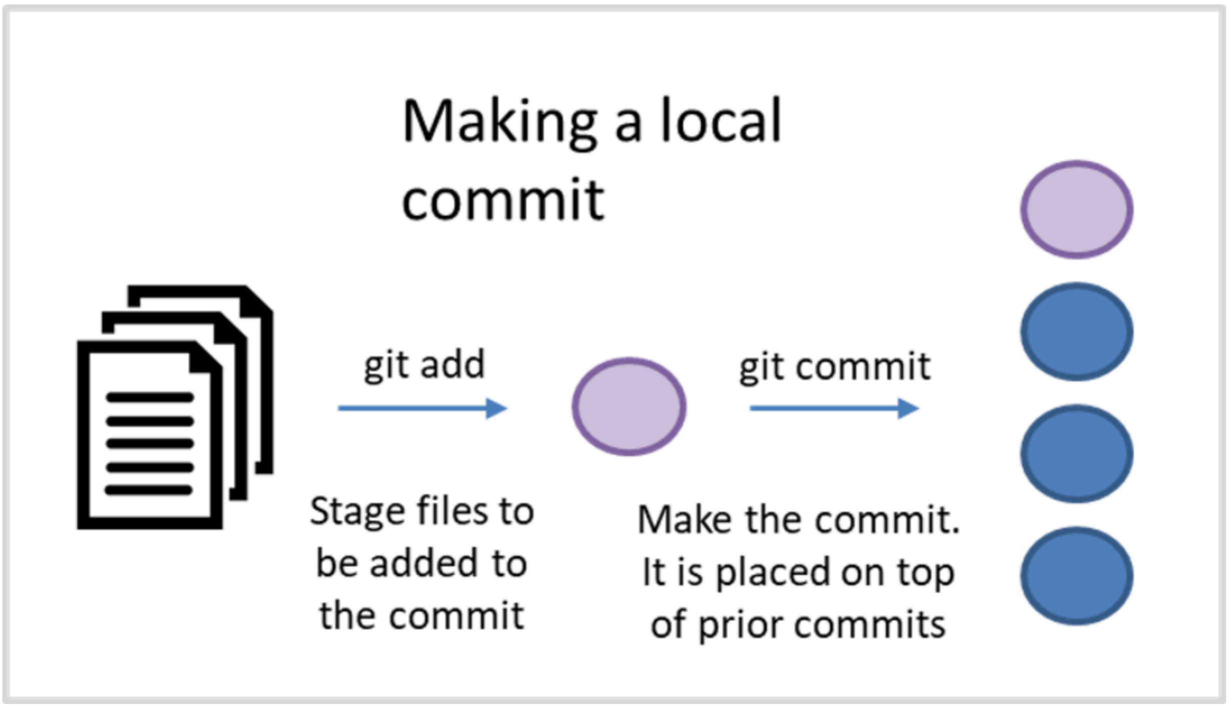
- stash - `git stash` temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on. Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

Git Workflow

- **Commit early and commit often** Make a commit after each small, meaningful change that adds, removes or modifies a single, isolated feature or function or to save your current progress. Think of a commit as a checkpoint in your work. When changing the code in small steps, it is also easier to see when and how you made a mistake in case something suddenly stops working.

- **Write meaningful commit messages** Just as you want the names of variables and functions in the program itself to be consistent and descriptive, you want the same for your commit messages. You want to understand what you have done when you later look at your old commits.

- **Keep your local and the remote repository in sync** Remember to pull updates made by your team members from the remote repository and to push your own changes so that the others can apply your changes into their local repositories. This is the only way you will get your team member’s updates and they will get yours.


Making a local commit
git add
Stage files to be added to the commit
git commit
Make the commit. It is placed on top of prior commits

Adding files to staging area

Before commiting a change, addition or removal you will first need to add the files to the staging area Whenever you make a change to your software it is a good idea to create a commit for your changes. The more incremental the change, the less that will be lost in the event that you need to revert to a previous commit.

```
git add . #adds all files that have been changed
# you can also add specific files
git add <file or deirectory> <another file or directory> ...
# for example:
git add main.py led.py
```

Once files are added to the staging area you should commit them Note that commits are only reflected in your local project.

```
git commit -m "<Descriptive message that describes what changes were mad
# for example:
git commit -m "Added led.py file, created a roatry encoder class"
```

Changing your last commit (if necessary)

In case you forgot to include some changes or a commit message in your last commit, you can stage the changes if needed (with the `git add` command and commit again using the `--amend` flag:

```
git add <forgotten file name>
git commit --amend -m "<amended message>"
```

Before being able to add your changes to the Remote project

Pulling updates from the central repository

Your local repository is completely isolated from the central repository. Changes made by others will not be applied to your local repository until you pull them from the central repository.

```
# Get updates made to the central repository
it pull --rebase origin main
# If there have been no changes pushed by your other team members
# you will get a message indicating that you are up to date with it
# this indicates that there is nothing more for you to do before you
# need to push your staged changes to the remote project
From https://gitlab.metropolia.fi/josephh/pico_project
 * branch            main       -> FETCH_HEAD
Current branch main is up to date.
```

With the `--rebase` option, the pull command adds all the new changes in the remote master on top of the local master and then places all local commits on top of that one by one. Without the --rebase option the remote changes and your local commits are merged into single merge commit.

Before you can run the git pull command you must either commit, discard or stash any local, uncommitted changes. Stashing changes for later use is explained in the next subchapter, ‘Stashing uncommitted changes’.

When pulling changes from a remote repository, you might face a merge conflict which must be handled before the updates can be applied. Merge conflicts and how to resolve them is discussed in its own section below.

Stashing Uncommited Changes

`Git pull` will fail in case you have any uncommitted, local changes. Before pulling updates to your local repository, you must either `commit` or `stash` those changes. In case you are not ready to `commit` your current work yet, you can `stash` your uncommitted changes for later use using the `git stash` command

Notice that we cannot pull due to unstaged changes

```
# With modified and and unstaged changes

git pull --rebase origin main
error: cannot pull with rebase: You have unstaged changes.
error: Please commit or stash them.

# With files staged , but not commited
git pull --rebase origin main
error: cannot pull with rebase: Your index contains uncommitted changes.
error: Please commit or stash them.
```

Git status confirms this:

```
# With modified and and unstaged changes

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:   test.py

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

# With files staged , but not commited

it status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.py
```

Now if `git stash` is used, it will shelf our changes and it we can work on things in another context without our current changes affecting things

```
git stash
Saved working directory and index state WIP on main: 88bd8d0 added joe.p
```

Now if we run `git status` we will see that it appears as if our changes have never happened.

```
git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Now we can pull from the remote repository, commit and push those changes, and then use `git stash pop` to return to your previous context

```
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:   test.py

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (fef8074a32852ead3fc41709836c695262629333)
```

And if we run `git status` again we see that we have been switched back to our original context

```
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:   test.py

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

Pulling changes from your remote project repository

Before you can push your changes to the remote Central repository of your project, you need to be sure that no changes have been pushed there before you. In this case you can first use the `git pull` command to try to pull the changes. If someone has updated the same thing that you have previously updated, you will be notified of merge conflicts. These will need to be fixed before you can proceed with your `pull` and ultimately your `push`

It is recommended to use the `pull` command with the `--rebase` flag as this will maintain all of the commits that your team members have made.

The best case scenario will provide you with an output like this, where there are no merge conflicts.

```
git pull --rebase origin main
From https://gitlab.metropolia.fi/josephh/pico_project
 * branch            main       -> FETCH_HEAD
Current branch main is up to date.
```

If there are no merge conflicts then you can just go ahead and `push` your changes to remote project repo.

```
git push origin main
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: 1
```

Now your local repository should be in sync with the remote project repo.

Resolving merge conflicts

Pulling updates from the remote master sometimes results in a merge conflict. This happens when there are changes to the same lines of code in both the updated version and your local commits. This may also happen when merging local branches together (More about branches in Chapter 7). When a merge conflict happens, you must choose which changes you want to keep.

```
git pull --rebase origin main
From https://gitlab.metropolia.fi/josephh/pico_project
 * branch            main       -> FETCH_HEAD
Auto-merging test.py
CONFLICT (content): Merge conflict in test.py
error: could not apply 147b3fc... Print added to test.py to cause a merg
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git r
Could not apply 147b3fc... Print added to test.py to cause a merge confl
```

`git status` will also show where your conflicts are located

```
git status
interactive rebase in progress; onto 31bc66a
Last command done (1 command done):
   pick 147b3fc Print added to test.py to cause a merge conflict
No commands remaining.
You are currently rebasing branch 'new_feature' on '31bc66a'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   test.py

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

As does `diff`

```
 sake of commenting,

++<<<<<<< HEAD
 +rb = fifo.Fifo(55)
 +print('I am test.py')
++=======
+ rb = fifo.Fifo(50)
+ print('I am test.py and I have been commited on a feature branch')
+ print('I am a print that has caused a conflict')
++>>>>>>> 147b3fc (Print added to test.py to cause a merge conflict)

  if rb.empty():
 +    print("Let's practice git stash")
      print('Fifo is empty')

  led = Pin(20, Pin.OUT)
```

Once you see where the conflict is, open the file in your editor or IDE git marks the conflicting lines in the file with the `<<<<<<< HEAD` marker. and then shows what the rremote changes are above the `=======` line and your local commit below. And it ends with `>>>>>>> <short commit hash> <commit message>` `+` indicates something added and `-` indicates something that has been removed.

In this case we do not have a conflict that will cause any harm, so we just remove the markers.

```
<<<<<<< HEAD
rb = fifo.Fifo(55)
print('I am test.py')
=======
rb = fifo.Fifo(50)
print('I am test.py and I have been commited on a feature branch')
print('I am a print that has caused a conflict')
>>>>>>> 147b3fc (Print added to test.py to cause a merge conflict)
```

And will be left with:

```
4 rb = fifo.Fifo(55)
print('I am test.py')
rb = fifo.Fifo(50)
print('I am test.py and I have been commited on a feature branch')
print('I am a print that has caused a conflict')
```

For the change to take effect, you must now save your changes in the editor and continue the rebasing. with the `--continue` flag. If more conflicts appear, fix them and `--contiue` rebasing ontil you have resolved all conflicts.

Once all conflicts are resolved, you can move on to pushing

4. Pushing local commits to the central repository

Once you have made a local `commit`, the changes must be pushed to the remote central repository so that other team members are able access them. Before pushing the changes, you must first make sure that your local repository is up to date with the current state of the central repository by first using `pull`.

```
it pull --rebase origin main
From https://gitlab.metropolia.fi/josephh/pico_project
 * branch            main       -> FETCH_HEAD
Already up to date.
```

If you get no merge conflicts it is safe to push your changes to the remote branch.

```
git push origin main
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), 432 bytes | 432.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
To https://gitlab.metropolia.fi/josephh/pico_project.git
   31bc66a..8f249fb  main -> main
```

Project history

The `git log --oneline` command can be used for viewing a brief version of the commit history of your local repository. The command lists the different commits you have made so far and your most recent commit shows at the top of the list. The log also shows the commits where your different branches point to. Each commit has a unique commit ID that is shown before the name of the commit:

```
git log --oneline

8f249fb (HEAD -> main, origin/main, new_feature) Print added to test.py
31bc66a Added a print output to test.py
88bd8d0 added joe.py and hotchkiss.py files
86b14f4 Modified the print message and the fifo size in test.py
cc3b4b2 Added comment to test.py demonstrate git functionality
ae24855 added message to print in test.py
7427c49 added a .gitignore file
9d1a38d Changed LED from the onboard PicoW LED to pin 20
e761c45 added comment to test.py
bb439a1 (old_origin/main, old_origin/HEAD) Pulled library update
3d3fec1 Updated lib
fa87a8a Added mqtt
cdf833d Changed wmic to taskkill and added title for local webserver
0120b45 Moved web server start earlier and added a delay before running
f7d4882 Moved web server start earlier and added a delay before running
bd47a39 Changed order of commands and display detected device name
2fa0d37 Changed order of commands, echo off, and display detected device
45445e9 Added more instructions
47e3a55 Added more instructions
1fd2e56 Added more instructions
c64d74d Added more instructions
7d41535 updated README
bf0763b Changed mpremote invoke style
c21adf5 Prevent MinGW path expansion during Pico update
06f765e Changed output wording
302cefd removed obsolete scripts
973c7a1 changed permissions on install.sh
ec14586 Added notification what pyhton executeable is used
bc66dd9 improved OSX python detection
6613789 Added python3.12 detection to OSX
4021478 Fixed missing file
7ab0475 renamed main.py to prevent automatic start at boot
9929e12 Changed shell script to work on git bash also
c9b8b97 Fetched updates
af9e1c1 Improved windows install script
cefa9eb Get port in OSX or Linux
0e7cf4b Added OSX/Linux install script
fef0e0d Added pico install script for windows
8ce238e Added pico-lib
1be5091 Initial commit
```

If you run the git log command without the --oneline option you will get a longer version of the log that includes additional information such as who made the commit and when. Furthermore, any commit messages with more than one line of text will be shown in full.

```
git log

commit 8f249fb075f7d02c77eab9bc3095402757dce834 (HEAD -> main, origin/ma
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Mon Nov 13 10:38:08 2023 +0200

    Print added to test.py to cause a merge conflict

commit 31bc66adb1601798c112c28425b97273cc5c51da
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Mon Nov 13 10:28:38 2023 +0200

    Added a print output to test.py

commit 88bd8d0fa183111334bc33019f3e07096c1b866f
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Sat Nov 11 19:50:40 2023 +0200

    added joe.py and hotchkiss.py files

commit 86b14f42d499252f8896c64d71bae42a2addafe3
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Sat Nov 11 17:01:08 2023 +0200

    Modified the print message and the fifo size in test.py

commit cc3b4b210c719ab6d225ff5f9eb8f0c62a362548
Author: josephh <joseph.hotchkiss@metropolia.fi>
Date:   Thu Nov 9 11:38:53 2023 +0200

    Added comment to test.py demonstrate git functionality
```

Viewing an old revision

Sometimes you want to take a look at how your repository looked at a specific commit without it affecting your existing work. You can do this by checking out the commit using its commit ID. Let’s say you wanted to take a look at how your code was when the “Changed message for motor controls” was the last made commit. As you can see from the image above, the commit ID for that commit is 973c7a1

```
git checkout 973c7a1
M       pico-lib
Note: switching to '973c7a1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in thi
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to f

HEAD is now at 973c7a1 changed permissions on install.sh
```

When you check out a specific commit this way, you end up in a ‘detached HEAD’ state. This means that the state of the repository you are looking at is not tied to any branch. You are able to make changes and even commit them without it affecting any of your existing branches. When you are ready, you can go back to your local master by issuing:

```
git checkout main
Previous HEAD position was 973c7a1 changed permissions on install.sh
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Undoing things

Sometimes you notice you have changed your code in an unwanted way and would like to go back in time where everything worked.

As you know, there are several phases in local changes:

- Local changes that have not been staged
- Local changes that have been staged with git add but not committed
- Changes that have been committed locally with git commit There as several ways of undoing changes depending on which phase you are with them.

Viewing and discarding unstaged changes

If you want to see how your current work in progress is different from your last commit, you can use the `git diff` command:

```
git diff

#outputs the diff to less

diff --git a/test.py b/test.py
index a9dd9a9..5f60564 100644
--- a/test.py
+++ b/test.py
@@ -2,14 +2,16 @@ import ssd1306
 import fifo
 import time
 from machine import Pin
+from led import Led

 import micropython
 micropython.alloc_emergency_exception_buf(200)
 # Comment for the sake of commenting,

-rb = fifo.Fifo(55)
 print('I am test.py')
-rb = fifo.Fifo(50)
+rb = fifo.Fifo(100)
+led1 = Led(22, brightness=5)
+led.on()
 print('I am test.py and I have been commited on a feature branch')
 print('I am a print that has caused a conflict')
```

In the image above, the lines with a minus sign at the beginning mark lines that have been removed since the last commit. Lines with a plus sign indicate added lines. If a line has been changed but not removed, the command will show both the old and new version as with the changed delay in the image above.

If you have made some local changes that you are not happy with and would like to start fresh from your last commit, you can discard the changes made to a file by running:

```
git restore <filename> # for one file
git restore . # for all files
```

Unstaging files

If you have changed files and already staged them with git add,

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.p

```

you can unstage them as follows:

```
git restore --staged test.py

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:   test.py

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

Undoing your last local commit

Before removing a commit, make sure that your problem is not solved by discarding local changes, changing your last commit with git commit --amend or making a new commit to fix whatever was wrong with the previous one. If none of these solves your problem and you are sure you want to remove your last commit, you can do so by resetting your HEAD to a prior commit.

```
# If you want to keep the changes in the commit
# the --soft flag will take your HEAD back one commit
# but keep the changes staged like they were before
# you committed them:

it reset --soft HEAD^

# If you run git log, you will see that you have gone
# back one commit. If you run git status, you will see
# your modified files show as staged
```

If you are 100% sure that you want to get rid of a commit for ever, then you can use the `--hard` flag

```
# If you are completely sure that you want to
# get rid of the last commit and the changes
# in it for good, you can use the --hard flag

git reset --hard HEAD^
HEAD is now at e83f614 Added variable for delay

# You can use git log to see that you have gone
# back one commit and verify there are no staged
# or unstaged changes running git status.
```