

Open-Source Software Practice

4. Git Misc. + Code Editor

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)

Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

Review: Git Advanced

- `git branch <branch_name>` creates a new branch based on HEAD.
- `git checkout <branch_name>` moves HEAD to a branch.
- `git merge <branch_name>` merges HEAD with a branch.
- `git fetch` fetches updates from a remote.
- `git push` pushes local branches to a remote.
- `git pull = git fetch + git merge`

In case of fire



1. `git commit`



2. `git push`



3. leave building

Show Commit Logs

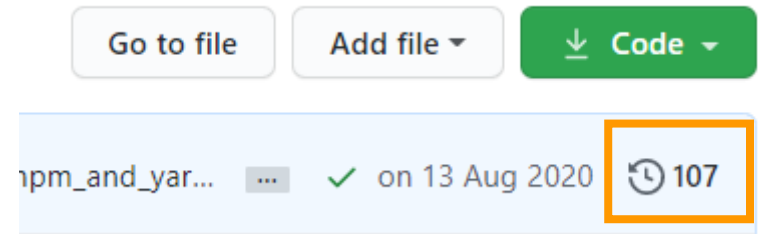
- `git log`
- Press 'q' to exit.
- But, I personally prefer to use the commit log page on GitHub.

```
$ git log
commit f3806f22001fdb1b5a906532f7933bc3e448b192 (HEAD -> main, origin/main, origin/HEAD)
Merge: f0a48b4 4cc8197
Author: Jaemin Jo <jmjo@skku.edu>
Date:   Fri Feb 5 12:57:50 2021 +0900

    Merge branch 'main' of github.com:e-/test

commit 4cc81975245b045c83ba14776c515710dc51db15
Author: Jaemin Jo <jmjo@hcil.snu.ac.kr>
Date:   Fri Feb 5 12:40:37 2021 +0900

    Update main.js
```



Tagging a Commit

- You can give a name to a commit.
- By default, each commit is assigned with an id, but this id is not human-readable.
- `git tag -a <tag_name> -m <message>`
- `git tag`

```
$ git commit
hint: Waiting for your editor to close the file...
[main f3806f2] Merge branch 'main' of github.com:e-/test
```

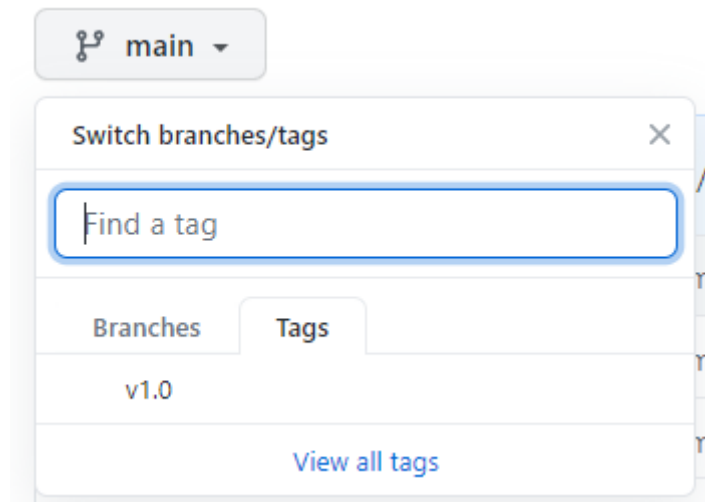
```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git tag -a v1.0 -m 'version 1.0'

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git tag
v1.0
```

Pushing a Tag

- To issue a tag, you must push it to a remote.
- But, `git push` does not push tags by default.
- Do it explicitly:
- `git push origin <tag_name>`

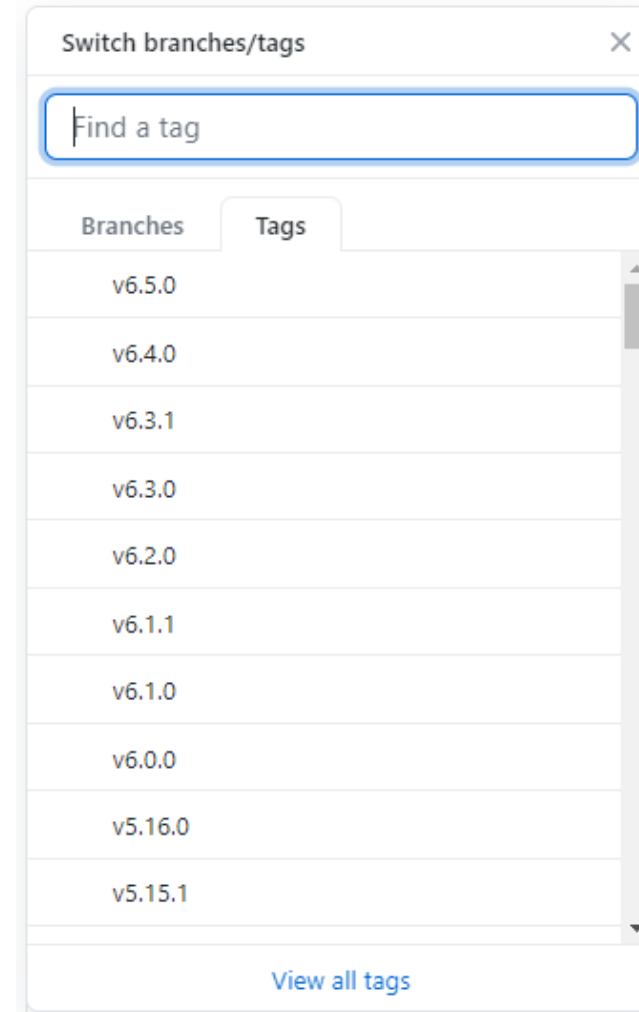
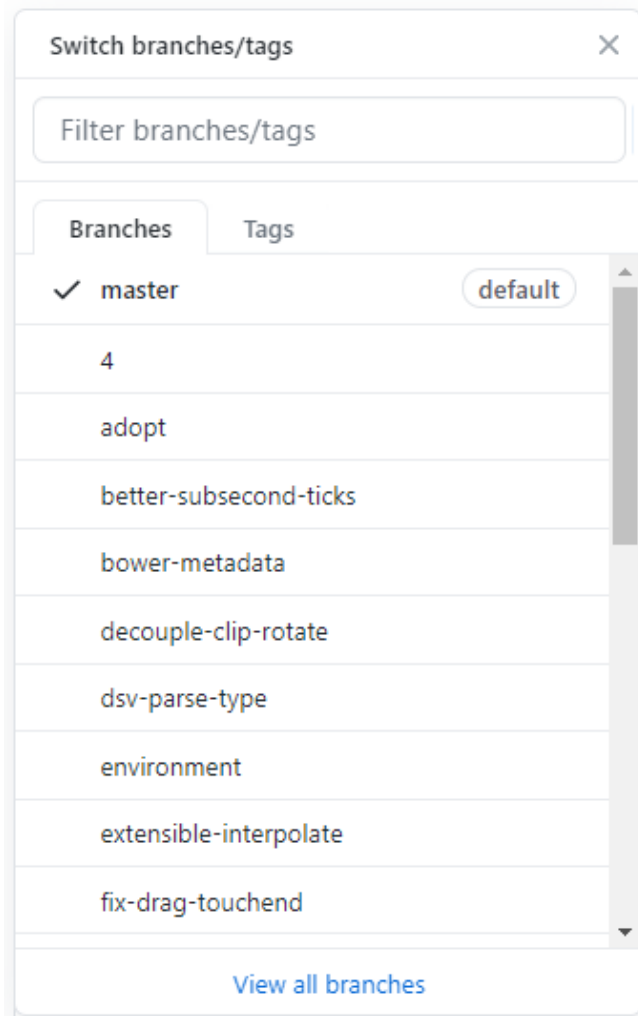
```
$ git push origin v1.0
Enter passphrase for key '/c/Users/jmjo/.ssh/id_rsa':
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 157 bytes | 157.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:e-/test.git
 * [new tag]          v1.0 -> v1.0
```



Branch vs Tag

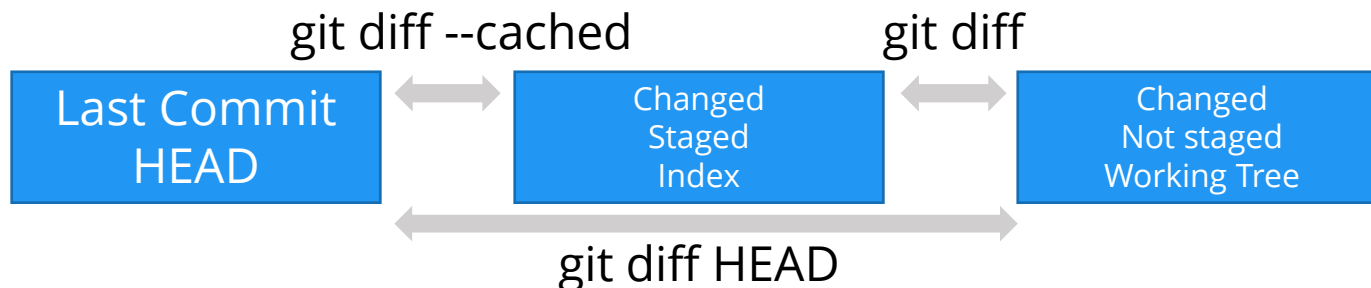
- **Branch:** a line of development
- **Tag:** an alias for a complex commit id
- You can checkout the commit that a branch or a tag refers to by running `git checkout <branch_name>/<tag_name>`
- A branch pointer **advances** each time you make a commit to that branch.
- A tag **never changes** unless you delete and create a new one.

Branch vs Tag



Comparing Code

- `git diff` shows changes between commits.
- `git diff` shows changes in the working tree not yet staged for the next commit.
- `git diff --cached` shows changes between the index and your last commit.
- `git diff HEAD` shows changes in the working tree since your last commit.



```
diff --git a/main.js b/main.js
index 356356a..ef08886 100644
--- a/main.js
+++ b/main.js
@@ -1,4 +1,5 @@
-let a, b, c, d;
-a = b + c;
+let a, b, d;
+a = b + c + 1;
+d = 1;
```


Stashing Changes

- Sometimes, you are interrupted in the middle of doing something.
 - Your boss comes to you and demands that you fix something immediately.
 - You forget to pull changes from a remote before you start a day.
- You need to temporarily store the changes you made and bring them back later.
- Making a commit? Not a good idea since your code is currently incomplete.

Stashing Changes

- `git stash` (store the current status)
- edit emergency fix
- `git commit -a -m "Fix in a hurry"`
- `git stash pop`

- `git stash`
- `git pull`
- `git stash pop` (merge conflicts can occur)

Two Remaining Important Topics

- git restore vs git reset
- git merge vs git rebase

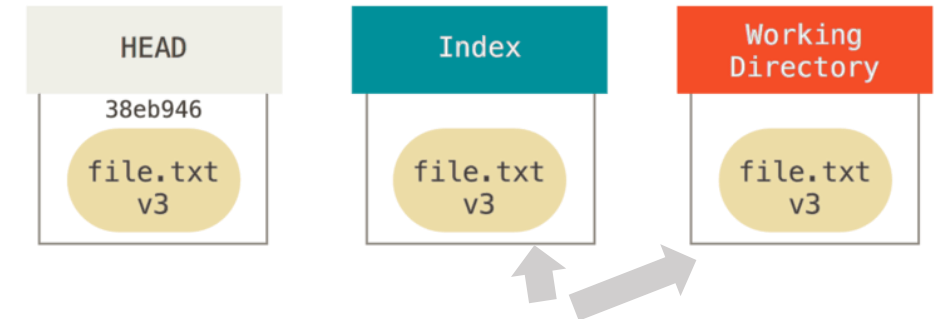
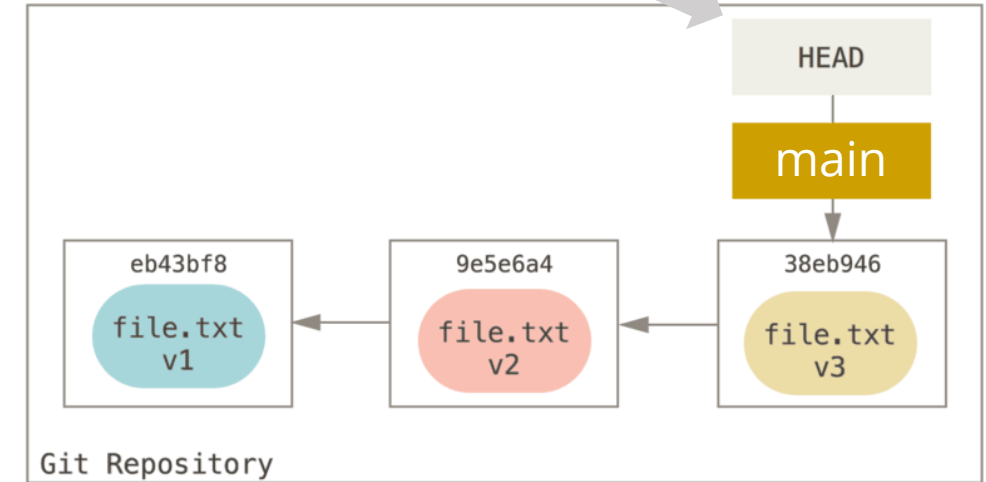
Git Reset

git reset changes HEAD,
but git restore does not.

IDCLab



- git reset changes the commit history.
 - Most important but most confusing command...
 - “What is the difference between reset and restore?” in a job interview
- git restore vs git reset
 - git restore does not move HEAD (does not change the history)
 - git restore --staged <file>
 - git restore <file> (discard changes)



git restore is only
interested in your index
(staged changes) and
working tree.

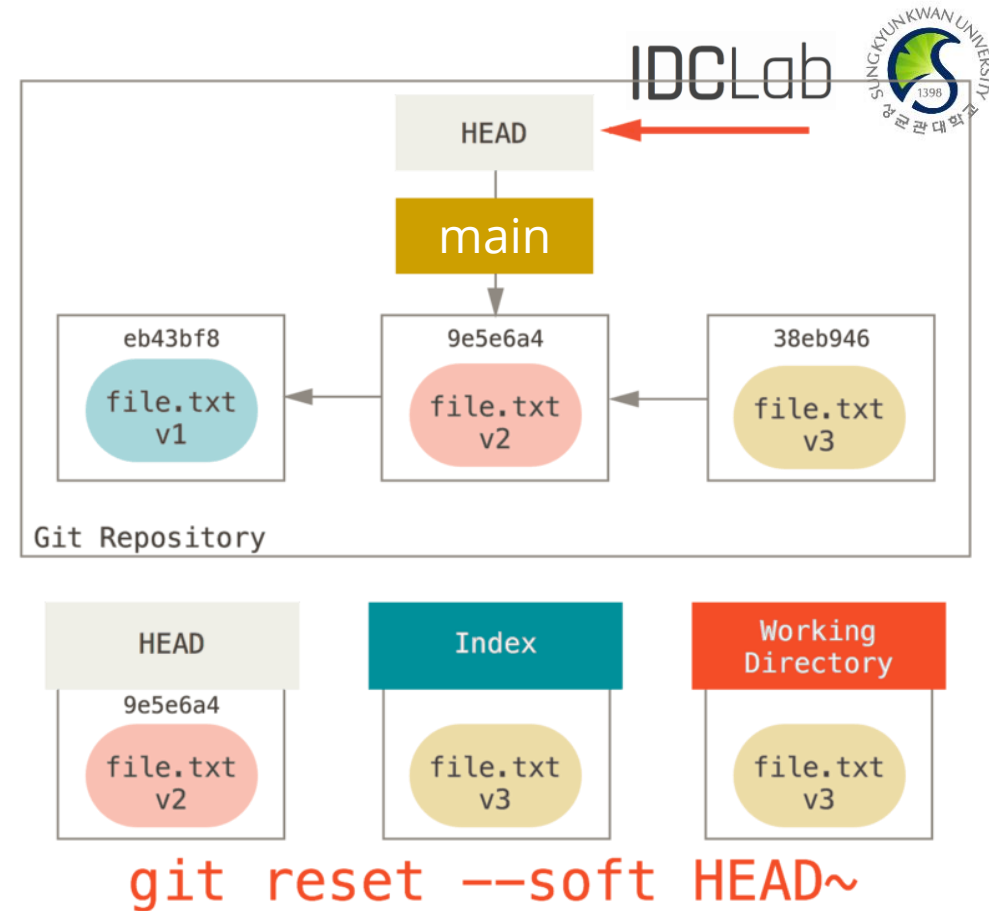
Git Reset

- `git reset --soft HEAD~`
- The soft reset only moves what HEAD points to.
- `HEAD~`: the parent of HEAD (= `HEAD~1`)
- `HEAD~2`: the parent of the parent of HEAD

```
jmo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git reset --soft HEAD~

jmo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ 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:   main.js
```



Git Reset

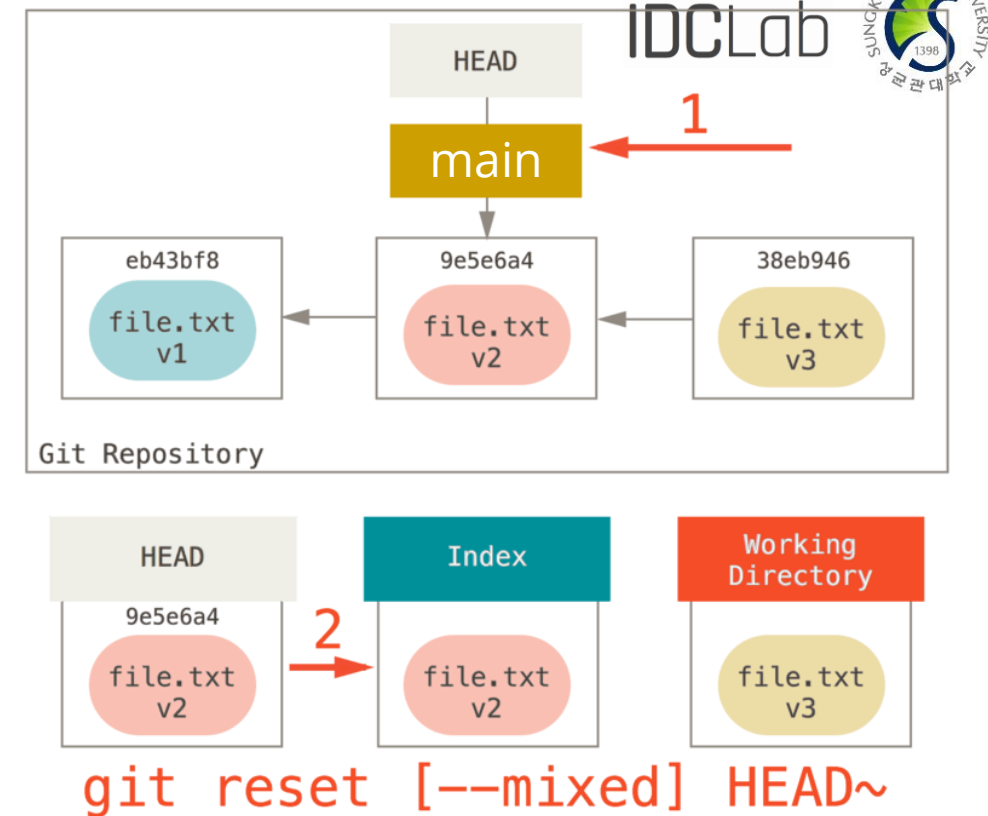
- `git reset HEAD~`
- The mixed reset moves what HEAD points to and updates the index as well.
- Useful when you committed something wrong, but you want to keep the changes to further modify.

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git reset --mixed HEAD~
Unstaged changes after reset:
M   main.js

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

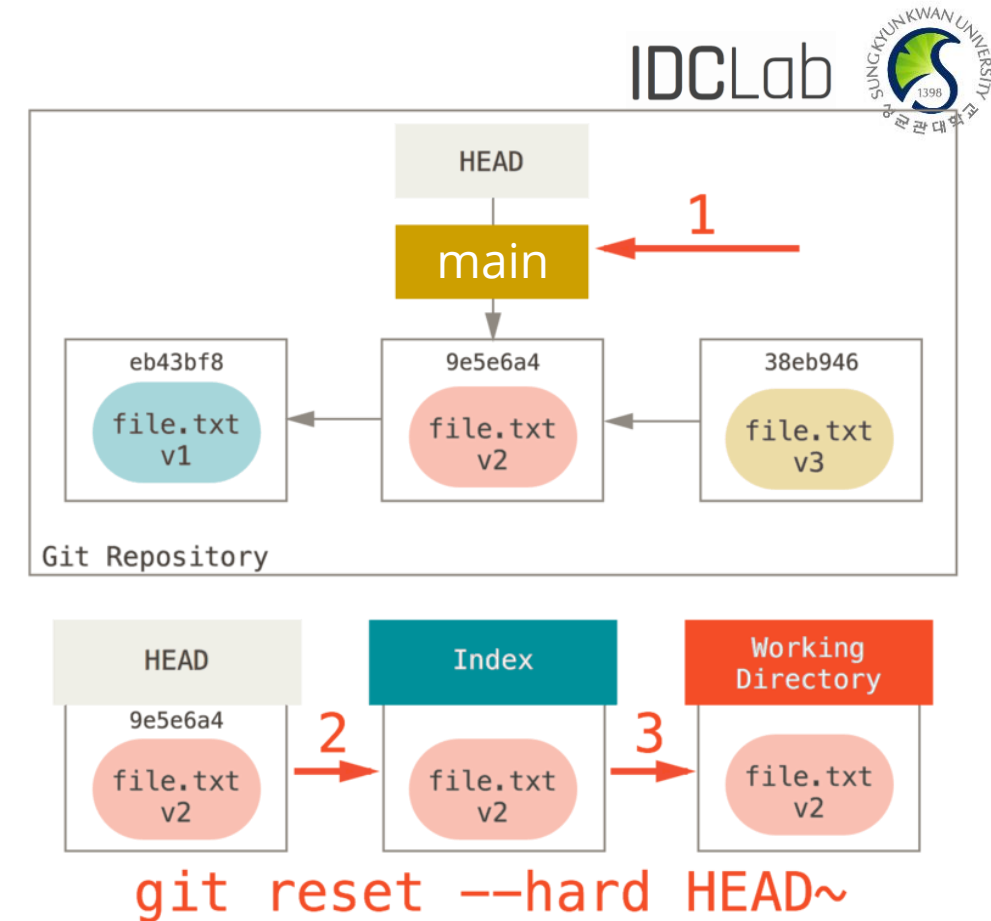
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.js

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



Git Reset

- `git reset --hard HEAD~`
- The hard reset moves what HEAD points to and updates the index and working tree as well.
- **YOU WILL LOSE CHANGES** in the working tree.
- Useful when you committed something *significantly* wrong and discard everything to start over.

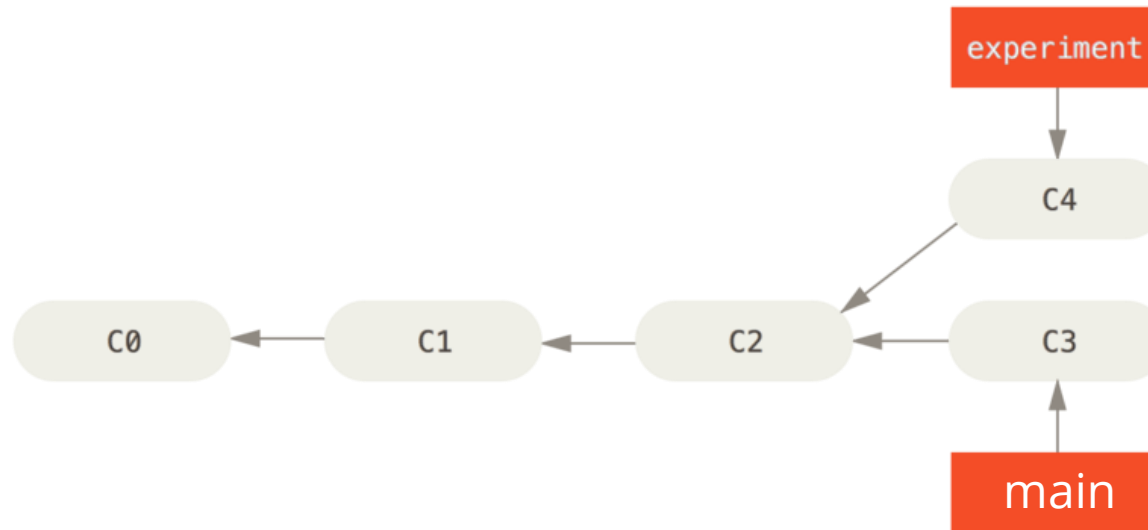


Let's Compare

- If you forget to add some files or want to edit the last commit message:
 - `git commit --amend`
- If you did something wrong but haven't committed them yet:
 - `git restore --staged <file>` (just unstage the file)
 - `git restore <file>` (unstage the file & discard changes)
- If you did something wrong and committed the changes:
 - `git reset --soft HEAD~` (rarely used)
 - `git reset HEAD~` ("Give me a second chance. I will modify and commit it again")
 - `git reset --hard HEAD~` ("I was totally wrong. Reset everything to the second last commit")

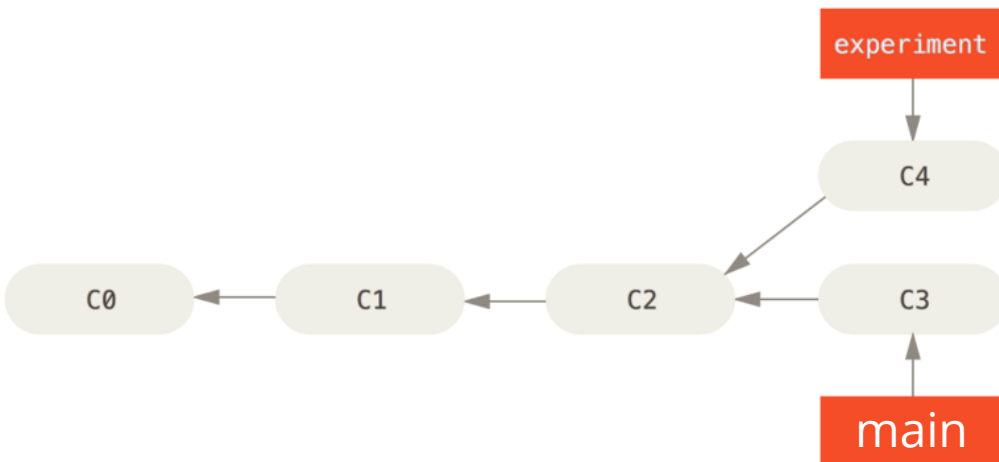
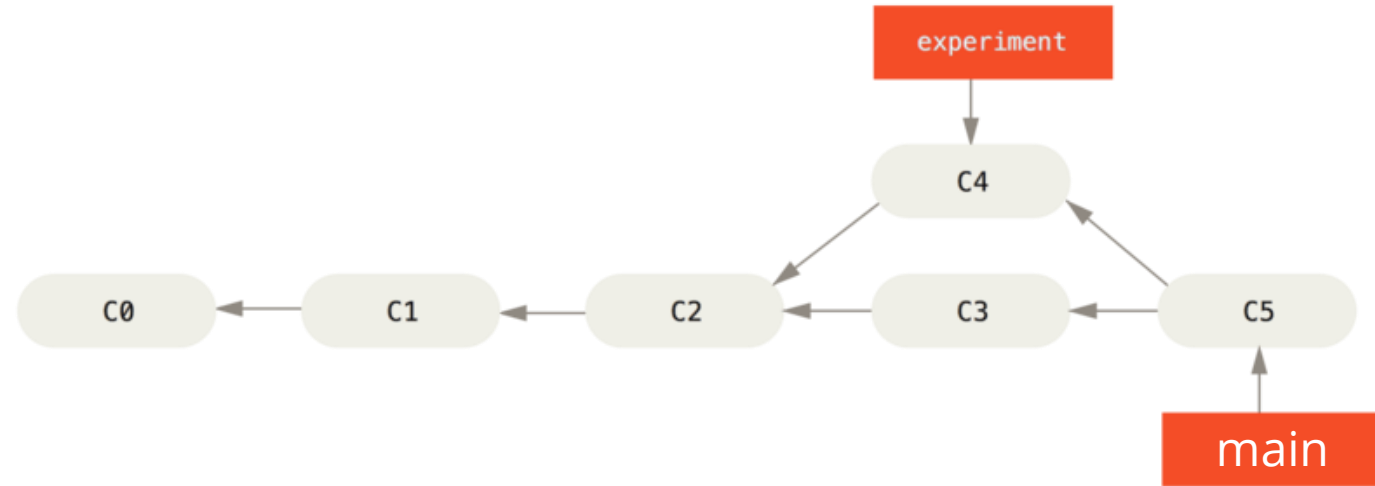
Git Rebase

- `git rebase` replays the changes committed to a branch on a different branch.
 - Another very important but confusing command.
 - “What is the difference between merge and rebase” in a job interview
- Both merge and rebase are methods to handle diverged commit history.



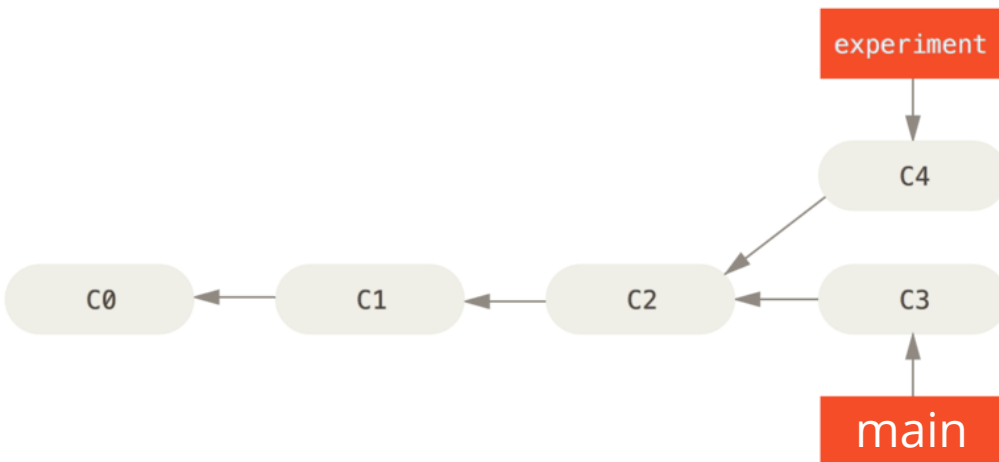
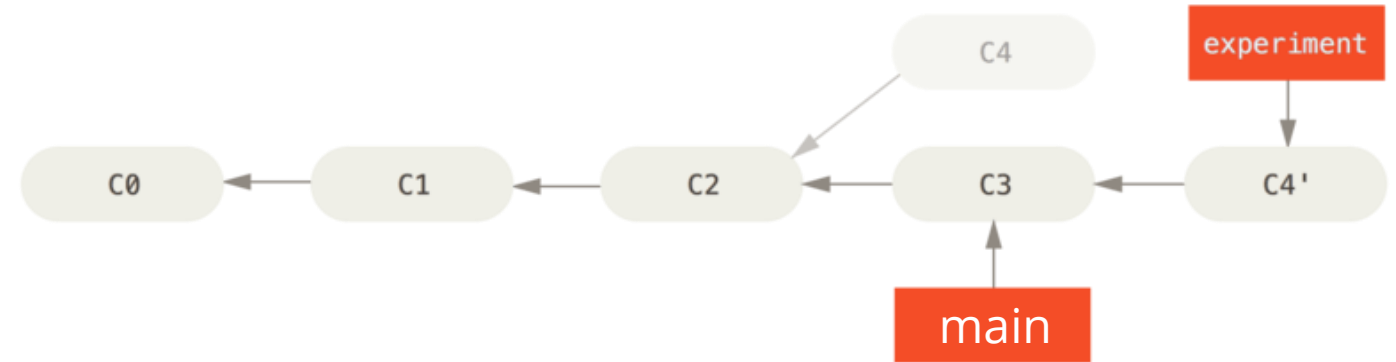
Git Merge

- `git checkout main`
- `git merge experiment`



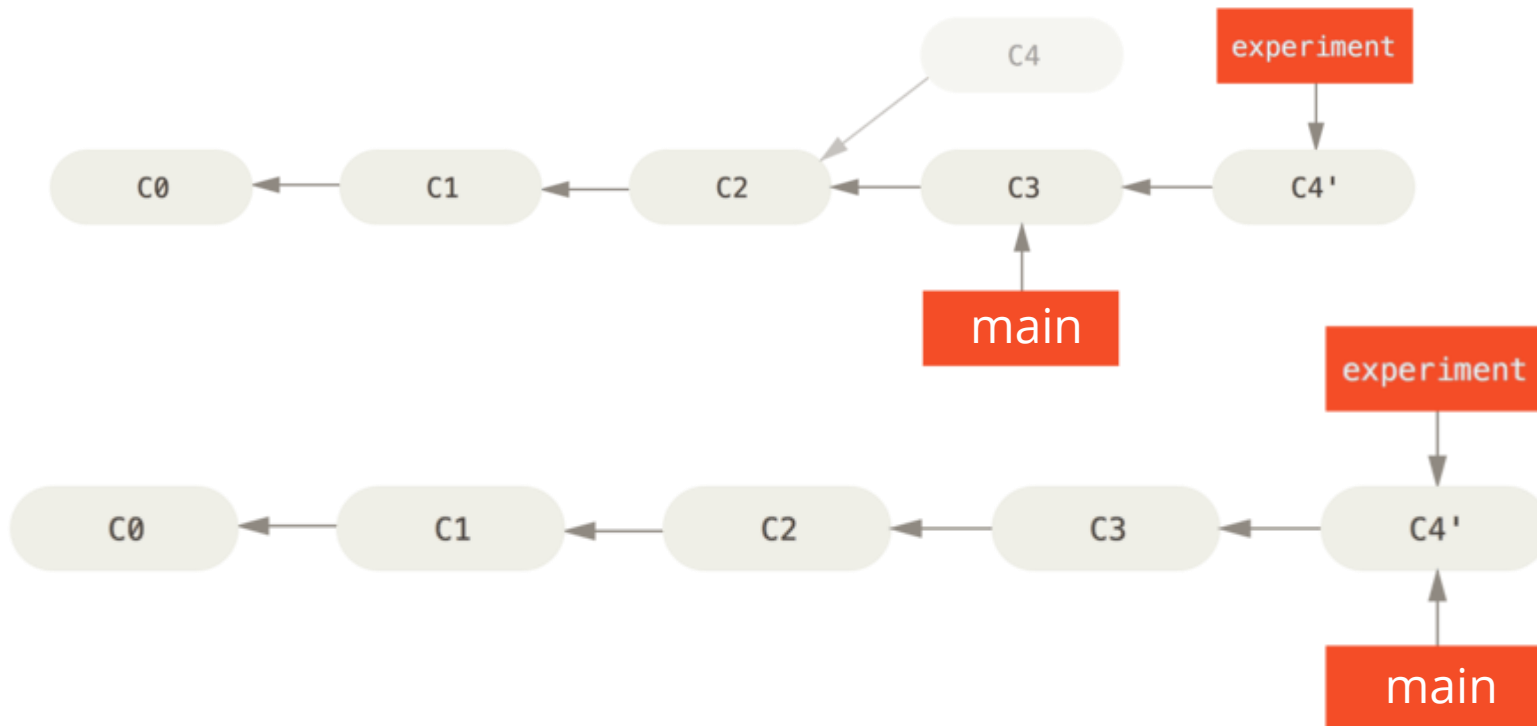
Git Rebase

- git checkout **experiment**
- git rebase main
- There can be conflict!

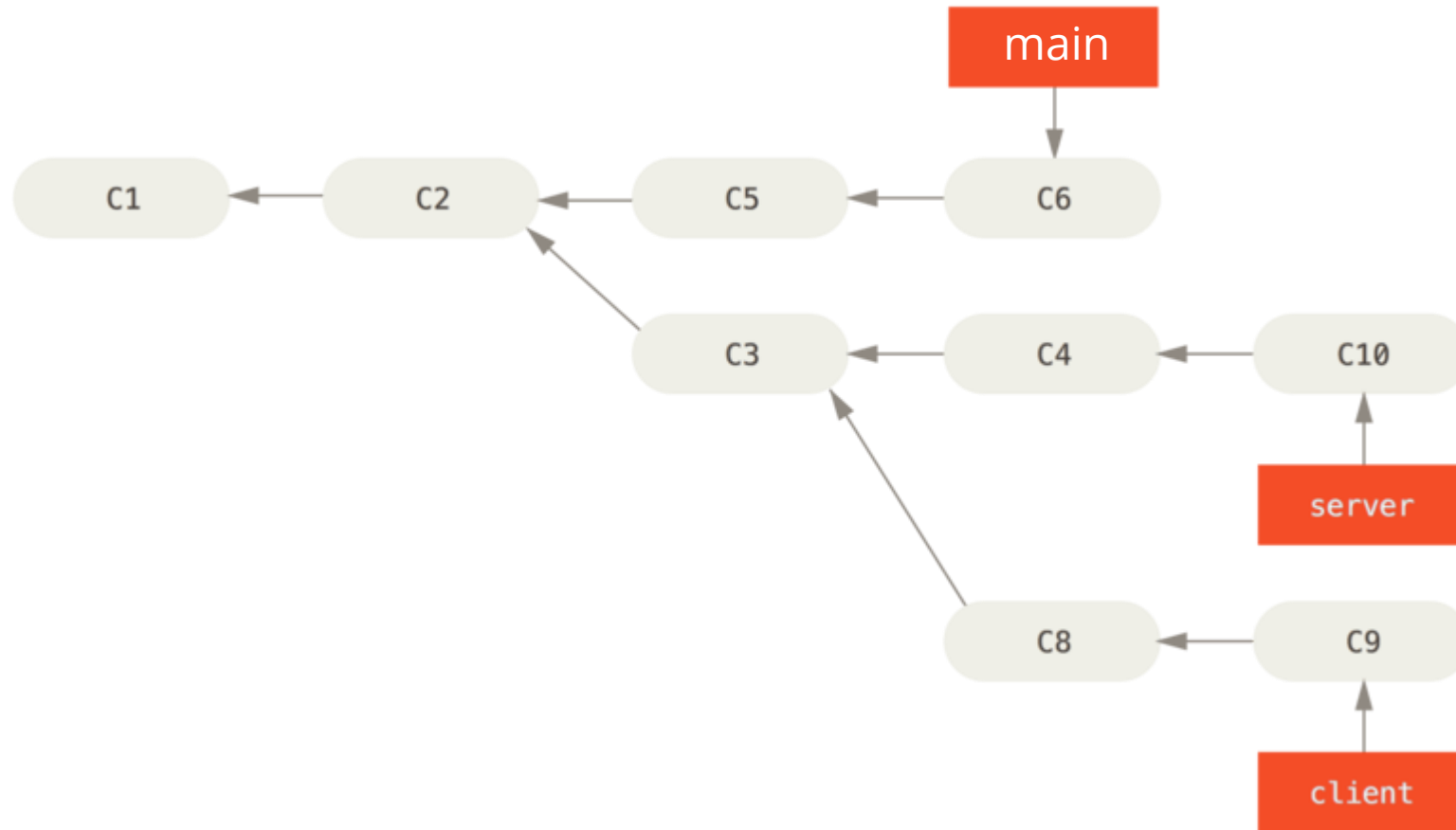


Git Rebase

- `git checkout main`
- `git merge experiment`

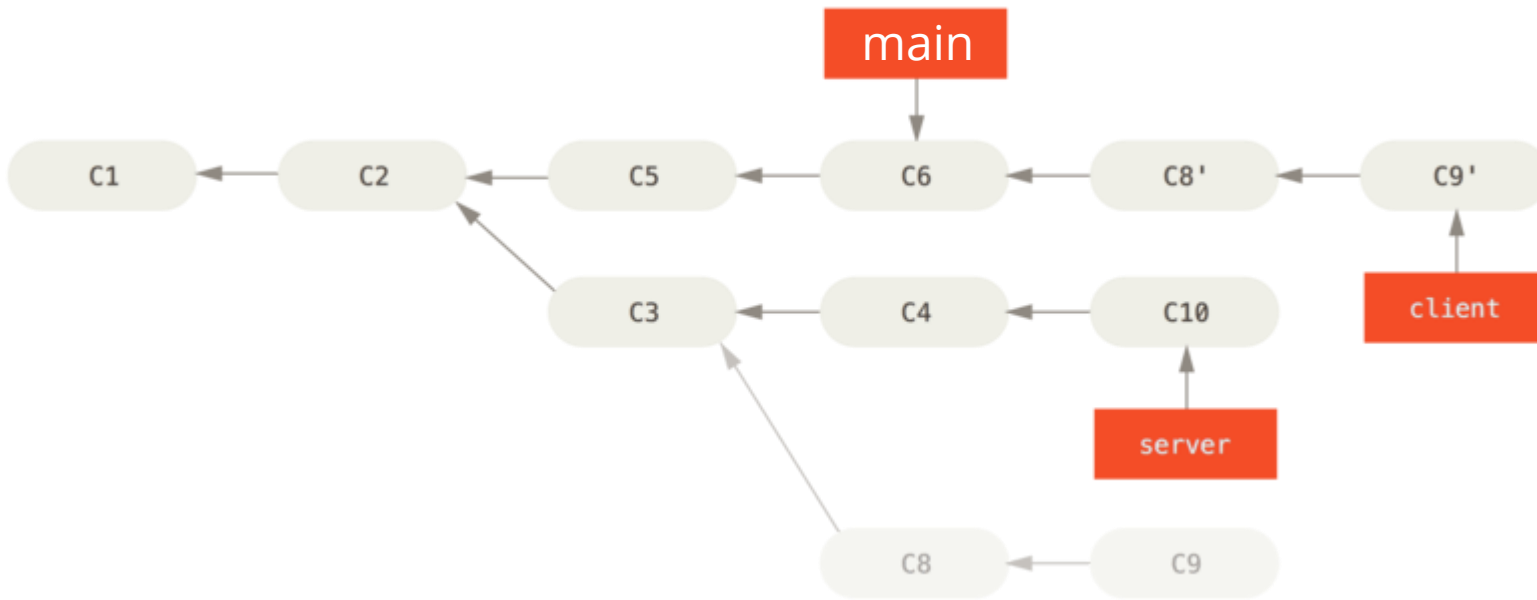


Git Rebase



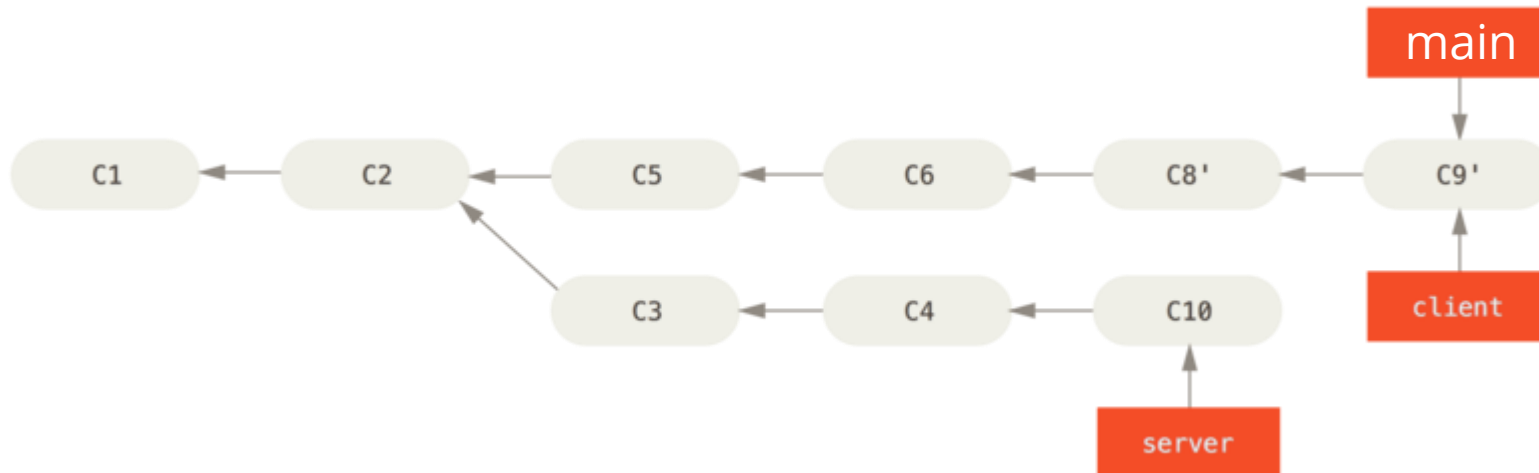
Git Rebase

- `git rebase --onto main server client`
- `git rebase --onto <newparent> <oldparent> <until>`
- `<until>` becomes the new HEAD.



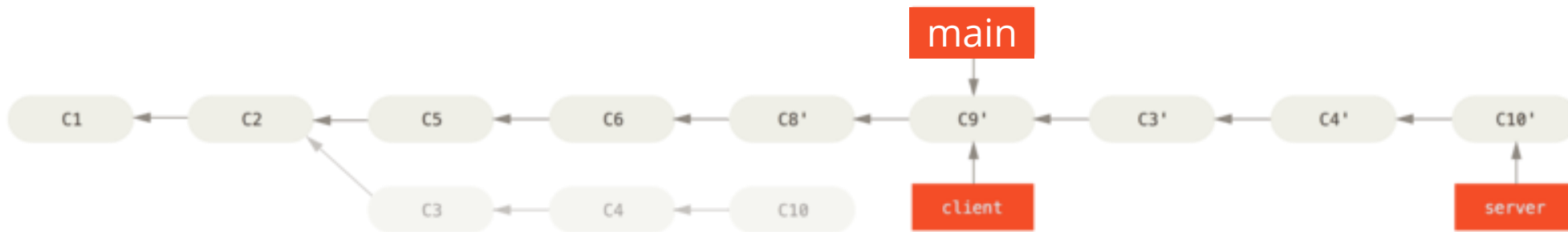
Git Rebase

- `git checkout main`
- `git merge client`



Git Rebase

- git checkout server
- git rebase main



Git Rebase

- `git checkout main`
- `git merge server`
- `git branch -d client`
- `git branch -d server`



Merge vs. Rebase

- Benefits of rebasing:
 - Makes the commit history **linear** as if only one developer worked.
 - Does not create **a merge commit**.
- Benefits of merging:
 - Preserves the complete commit history.
 - Preserves the context of branches.
- See how your colleagues do.
- A good article: <https://www.atlassian.com/git/articles/git-team-workflows-merge-or-rebase>

Git Rebase

- Rebasing cannot be used to avoid merge conflicts that would occur during merging.
- Resolve conflicts and `git rebase --continue`.

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (exp)
$ git rebase main
error: could not apply 00f9eee... wer
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort"
.
Could not apply 00f9eee... wer
Auto-merging main.js
CONFLICT (content): Merge conflict in main.js

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (exp|REBASE 1/1)
$ git status
interactive rebase in progress; onto f8f9826
Last command done (1 command done):
  pick 00f9eee wer
No commands remaining.
You are currently rebasing branch 'exp' on 'f8f9826'.
  (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:   main.js

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

Finding the Right Setting

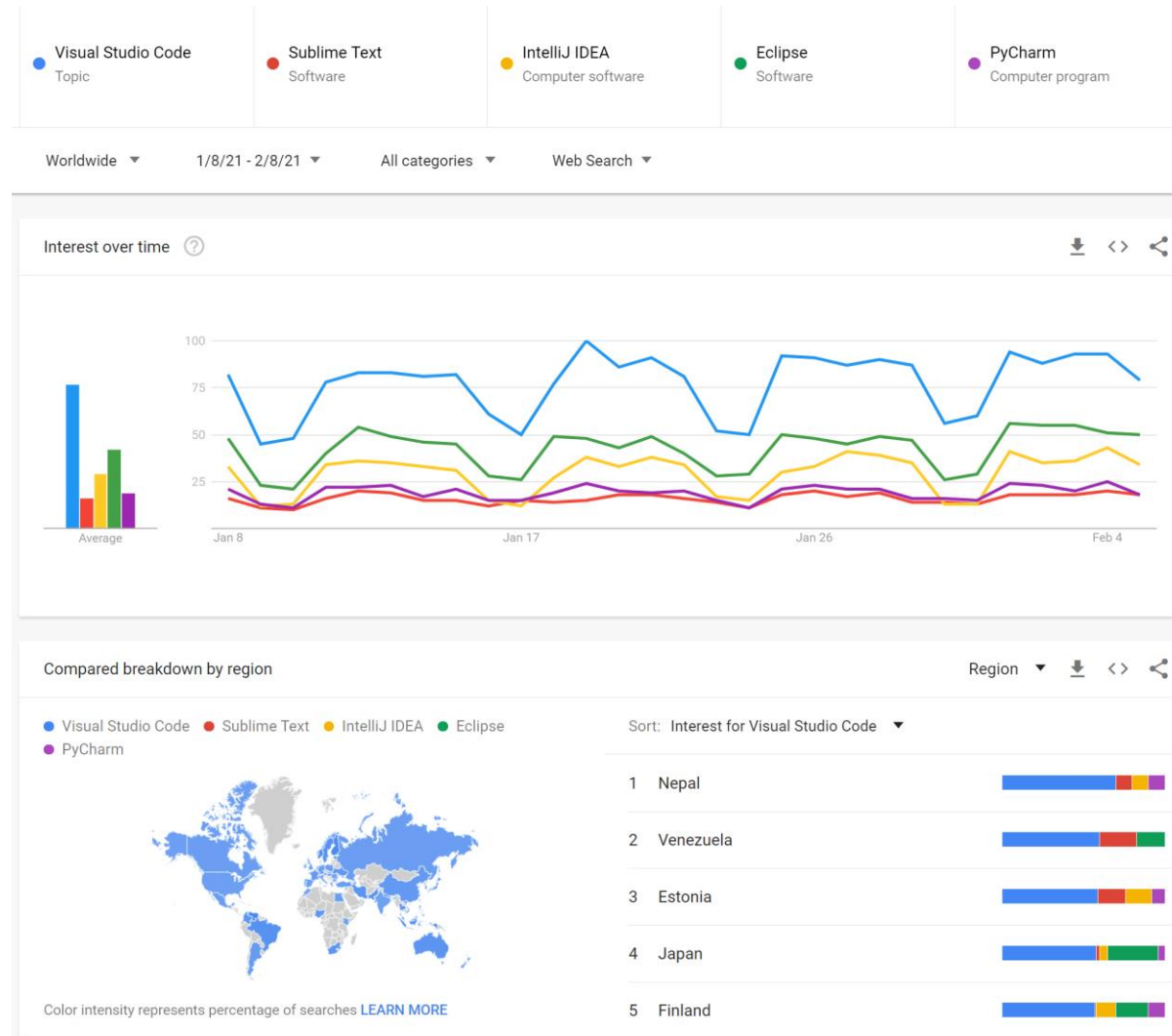
- Setting up your development environment (hardware + software) is crucial in your career.
 - Time
 - Productivity
 - Less pain (ergonomics)
- Investing 1 hour to making your own setup can save 100 hours in the future!
 - Trial and error
 - A bad workman blames his tools, but a pro does care about the tools!

Finding the Right Setting

- Hardware settings
 - Monitors / layout
 - Keyboard
 - Mouse
 - ...
- Software settings
 - OS
 - **Code editor**
 - Shortcuts
 - ...

- In your career as a software engineer, you would spend the most time with a code editor.
- What is a good code editor?
 - Free
 - Light-weight
 - Extension (features)
 - Maintained
 - Reliable
 - Scalable
 - Community

Searching Trends



Code Editor

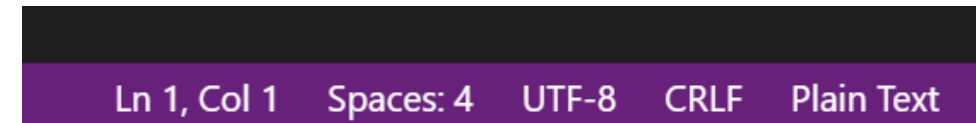
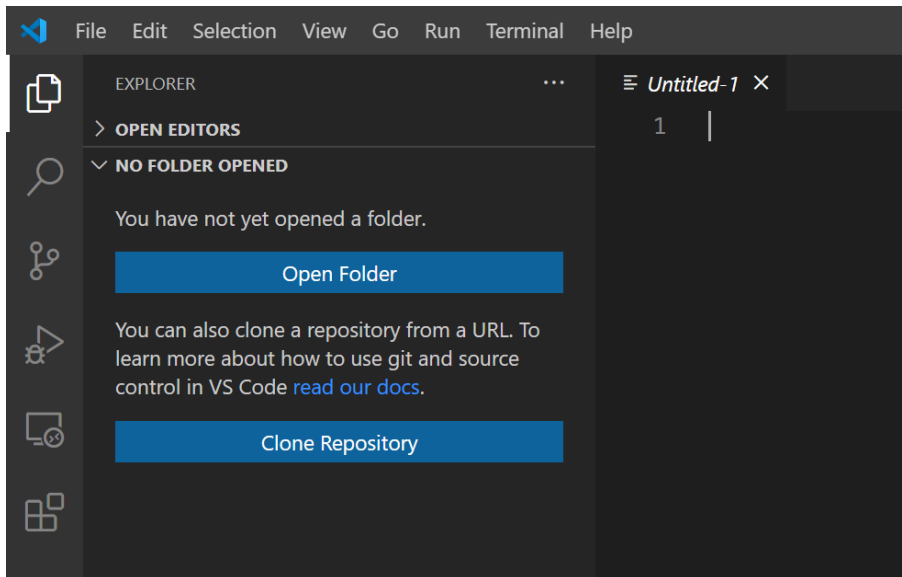
- IDEs and editors that I have used:
 - Visual Studio (+15 years)
 - Vim (+8 years)
 - Visual Studio Code (+6 years)
 - Sublime Text (3 years)
 - Notepad++ (6 months)
 - Atom (3 months)
- Finally, I settled down with this setting:
 - **Visual Studio Code** for most projects (JavaScript, Python, Web, Markdown, LaTeX, ...)
 - **Visual Studio** for C and C++ projects
 - **Vim** for simple editing on Mac/Linux (configuration files)

Prerequisite

- Install VSCode (<https://code.visualstudio.com/>)
 - I strongly recommend setting the language to English.

Visual Studio Code

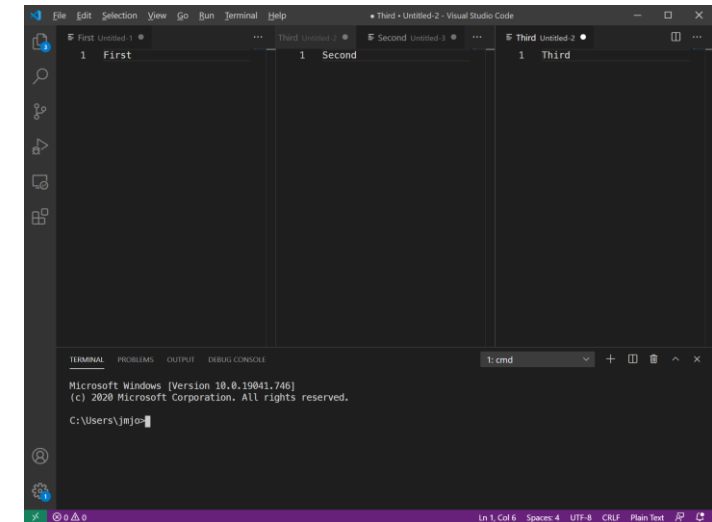
- Open the root directory of your project.



Make sure the encoding is set to UTF-8 if you want to work on files in different OSes.
We will learn how to automate this later.

Keyboard Shortcuts

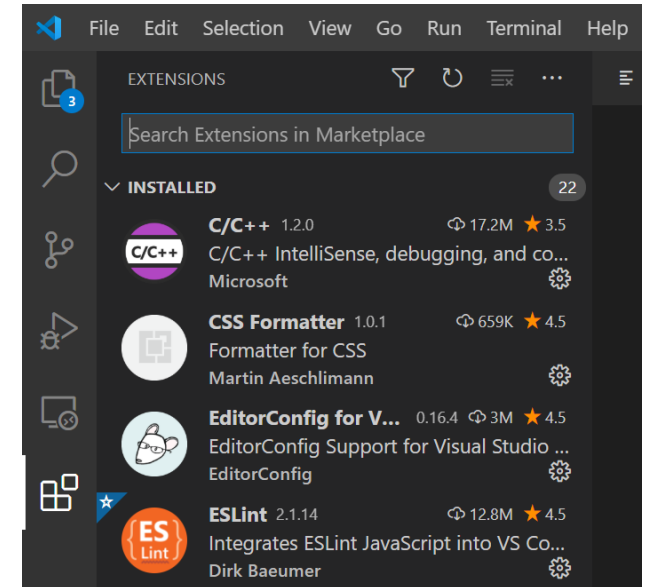
- General
 - Command palette: Ctrl + Shift + P
 - Quick open: Ctrl + P (Useful when you open a file in the project directory)
 - New file: Ctrl + N
 - Save: Ctrl + S
 - Delete one line: Shift + Del
- Side-by-side editing
 - Split: Ctrl + \ (backslash)
 - Close the current tab: Ctrl + W
 - Move the current file to the next/previous tab: Ctrl + Alt + Left or Right
 - Switch between tabs: Ctrl + 1, Ctrl + 2, Ctrl + 3
- Refactoring
 - Rename a symbol: F2
 - Auto-format the document: Shift + Alt + F
 - Go to definition: F12
- <https://code.visualstudio.com/docs/getstarted/tips-and-tricks>



Must-Use Plugins

- EditorConfig ([EditorConfig](#))
- ~~Code formatter ([Prettier](#))~~
- IntelliCode ([Microsoft IntelliCode](#))
- Language-specific helper
 - Javascript: built-in support (👍)
 - Python: [Microsoft Python](#)
- Git (built-in support, 👍)

- There are many plugins other than these. Explore the marketplace by yourself.



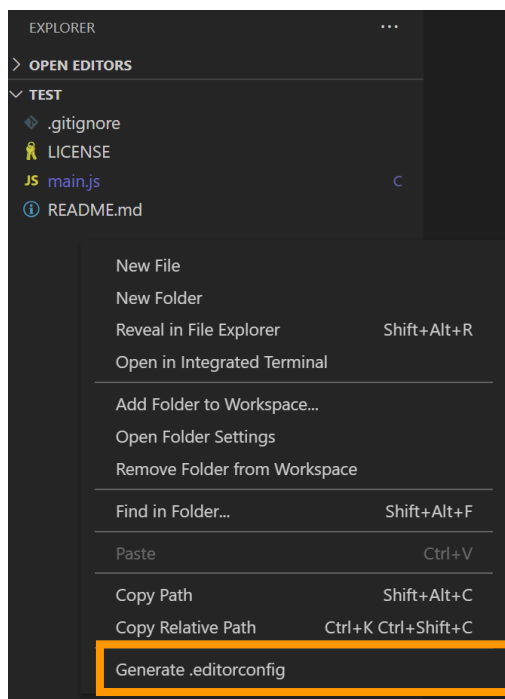
EditorConfig



- Why EditorConfig?
 - To maintain consistent coding styles for multiple developers working on the same project across various editors and IDEs.
 - Tab for indent, Tab size: 4, utf-8
 - Space for indent, Tab size: 2, utf-8
 - Space for indent, Tab size: 4, euc-kr
- Specify end of line, character set, indent style, indent size, and tab width used throughout a project.
- Official: <https://editorconfig.org/>

EditorConfig

- Place an “.editorconfig” file under the root directory.



```
# top-most EditorConfig file
root = true

[*]
indent_style = space
indent_size = 4
end_of_line = crlf
charset = utf-8
trim_trailing_whitespace = false
insert_final_newline = false
```

```
# EditorConfig is awesome: https://EditorConfig.org

# top-most EditorConfig file
root = true

# Unix-style newlines with a newline ending every file
[*]
end_of_line = lf
insert_final_newline = true

# Matches multiple files with brace expansion notation
# Set default charset
[*.{js,py}]
charset = utf-8

# 4 space indentation
[*.{py}]
indent_style = space
indent_size = 4

# Tab indentation (no size specified)
[Makefile]
indent_style = tab

# Indentation override for all JS under lib directory
[lib/**/*.js]
indent_style = space
indent_size = 2

# Matches the exact files either package.json or .travis.yml
[{package.json,.travis.yml}]
indent_style = space
indent_size = 2
```

EditorConfig

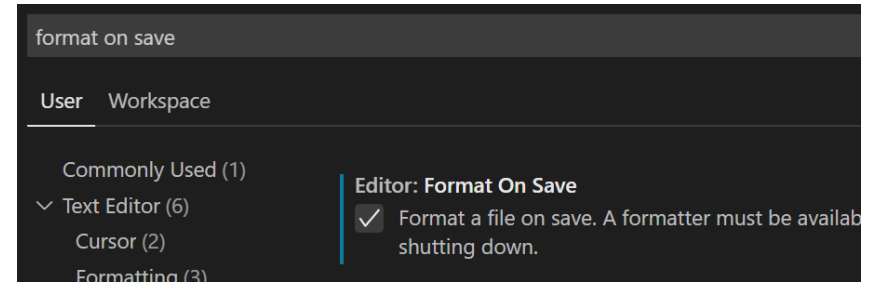
- You must add `.editorconfig` as part of your project.
 - `git add .editorconfig`
- You can use any other code editor instead of VSCode, but the editor must support EditorConfig.
 - Most editors have their own plugins.
 - Code editors without EditorConfig support are not for collaboration. Never use them.
- Take a look at `.editorconfig` used in Airbnb
 - <https://github.com/airbnb/javascript/blob/master/.editorconfig>

Code Formatter

- Since March 2022, Microsoft has added built-in support for most front-end languages (Javascript, CSS, HTML), you don't have to install extra formatting plugins anymore.
- <https://www.roboleary.net/tools/2022/05/18/vscode-you-dont-need-a-formatting-extension-prettier-and-friends.html>

Setting up Formatter

- File -> Preference -> Settings
- Turn on Format on Save:
 - Search for “format on save”
 - Make sure the checkbox checked



Formatter

```
1  for(let i=0;i<10;i++){  
2    |    if(i%2 ===0)console.log(i);  
3  }
```



Ctrl + S

```
1  ✓ for (let i = 0; i < 10; i++) {  
2    |    if (i % 2 === 0) console.log(i);  
3  }
```

Formatter vs. Linter

- **A formatter** only formats code and does not care its semantics.
 - e.g., Prettier
 - Maximum number of characters on a line
 - Tab size
 - Comma styles
 - ...
- **A linter** does more and is likely to catch real bugs.
 - e.g., ESLint
 - No unused variables
 - No implicit global variables
 - ...

Visual Studio IntelliCode

- “The Visual Studio IntelliCode extension provides AI-assisted development features for Python, TypeScript/JavaScript and Java developers in Visual Studio Code, with insights based on understanding your code context combined with machine learning.”
- Official:
<https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.vscodeintellicode>
- I recommend using it. Install it from the marketplace.

Visual Studio IntelliCode

```
5 let array = [1, 2, 3, 4];
6
7 array.
```

- ★ push
- ★ map
- ★ forEach
- ★ length
- ★ join
- concat
- copyWithin
- entries
- every
- fill
- filter
- find

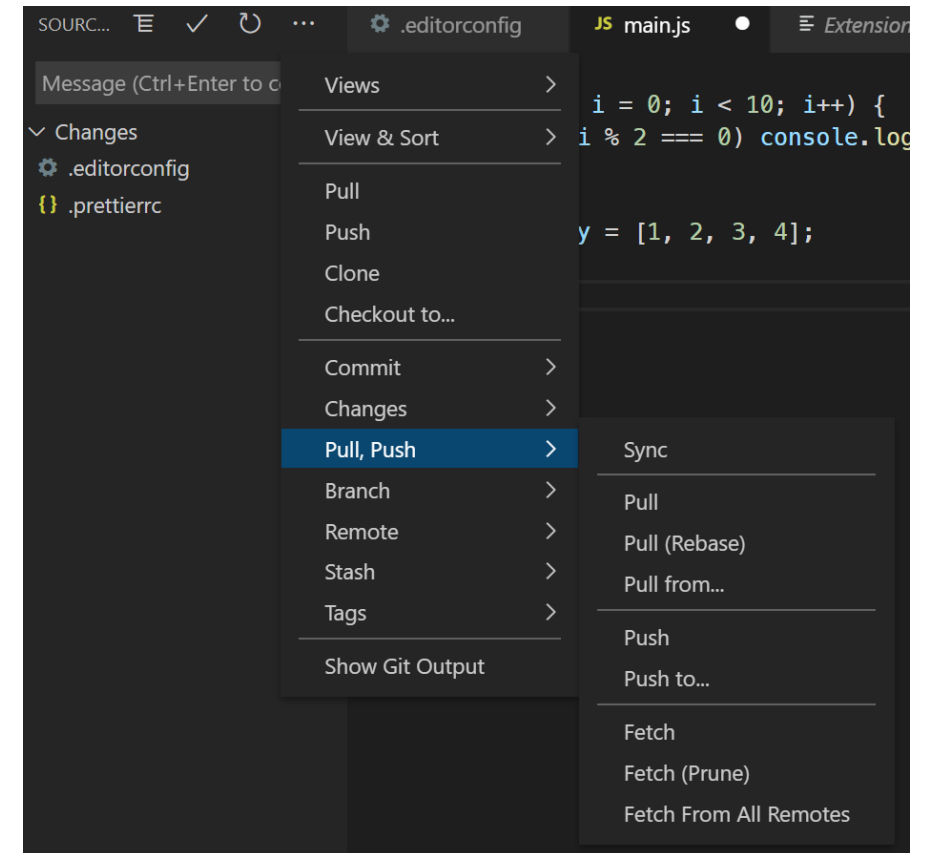
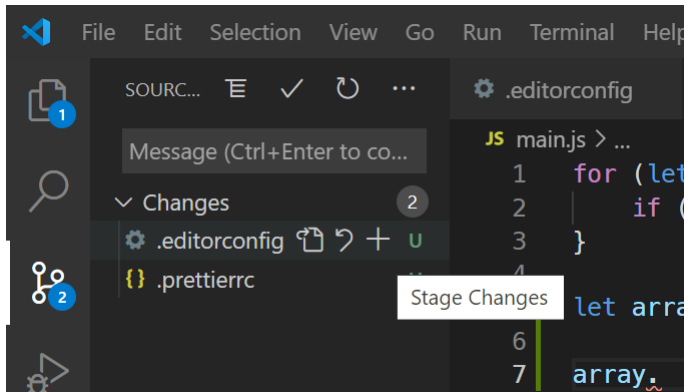
(method) Array<number>.push(...items: number[]): number

Appends new elements to an array, and returns the new length of the array.

@param items — New elements of the Array.

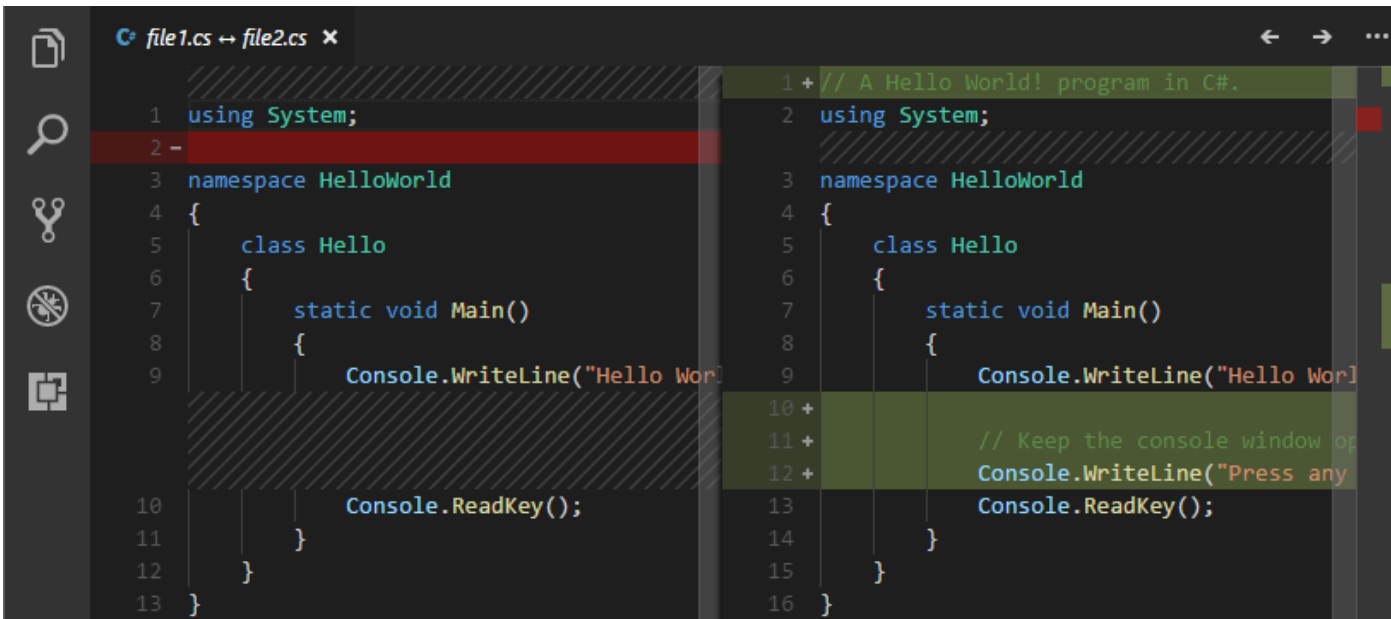
Git Support

- Shortcut: Ctrl + Shift + G (G for **G**it)
- Click “+” to stage (git add).
- Type a message and Press Ctrl + Enter to commit (git commit).

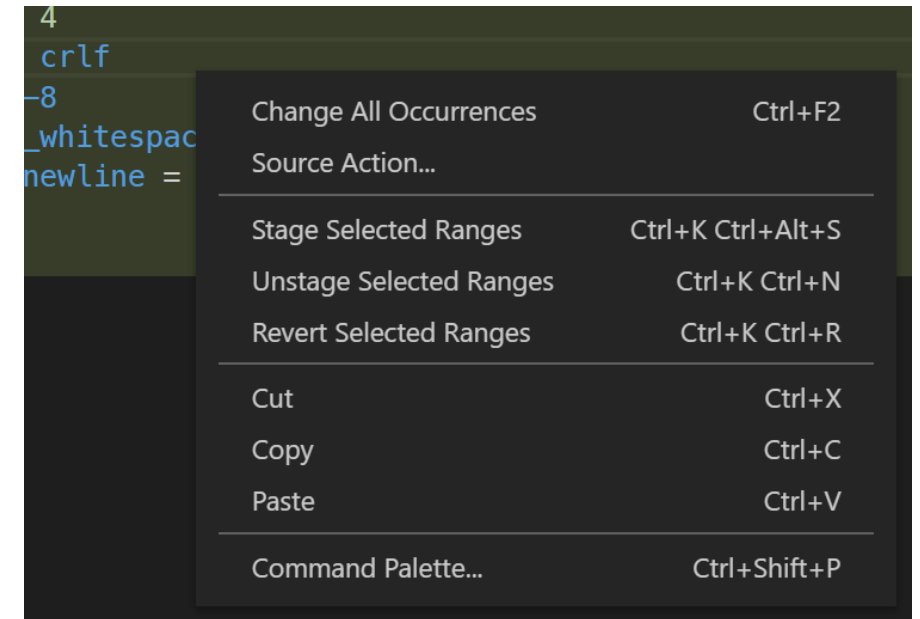


Git Support

- Click on a file on the source control panel to diff.
- You can even revert some changes (super convenient).



```
1 using System;
2 -
3 namespace HelloWorld
4 {
5     class Hello
6     {
7         static void Main()
8         {
9             Console.WriteLine("Hello World");
10
11             Console.ReadKey();
12         }
13     }
14 }
15
16 // A Hello World! program in C#.
17 using System;
18 namespace HelloWorld
19 {
20     class Hello
21     {
22         static void Main()
23         {
24             Console.WriteLine("Hello World");
25
26             // Keep the console window open
27             Console.WriteLine("Press any key to continue.");
28             Console.ReadKey();
29         }
30     }
31 }
```



Change All Occurrences	Ctrl+F2
Source Action...	
Stage Selected Ranges	Ctrl+K Ctrl+Alt+S
Unstage Selected Ranges	Ctrl+K Ctrl+N
Revert Selected Ranges	Ctrl+K Ctrl+R
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Command Palette...	Ctrl+Shift+P

Git Support

- Visual support for resolving merge conflicts

```
$ git push
Enter passphrase for key '/c/Users/jmjo/.ssh/id_rsa':
To github.com:e-/test.git
! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'github.com:e-/test.git'
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.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
let a, b;
=====
let c, d;
>>>>>> 4cc81975245b045c83ba14776c515710dc51db15 (Incoming Change)
```


Summary: Git Misc. + Code Editor

- `git tag / git diff`
- `git stash/pop` store/pop changes temporarily.
- `git reset` changes HEAD! (soft, mixed, and hard)
- `git rebase` moves commits to a different branch as if they happened on that branch.
- VS Code + EditorConfig + IntelliCode + Git = Lv. 1 gear (🔧🛡️) of a software engineer

