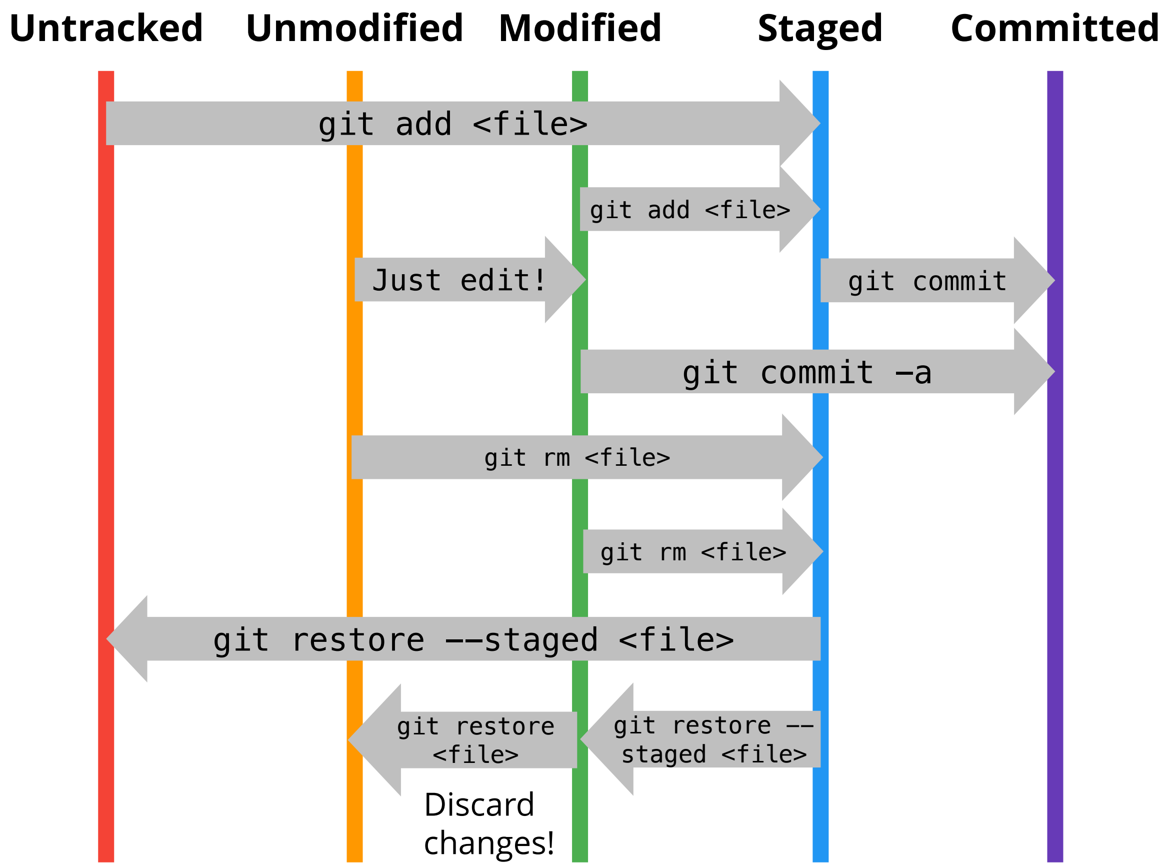
Open-Source Software Practice

3. Git Advanced

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

Review: Git Basics



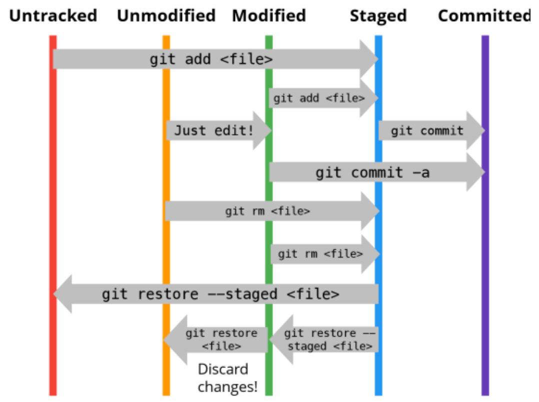
Working with Remotes

- Commands for today: branch, checkout, merge, fetch, push, and pull

- These commands work on *commits* not *individual files*.
  - A commit can have changes made on multiple files.

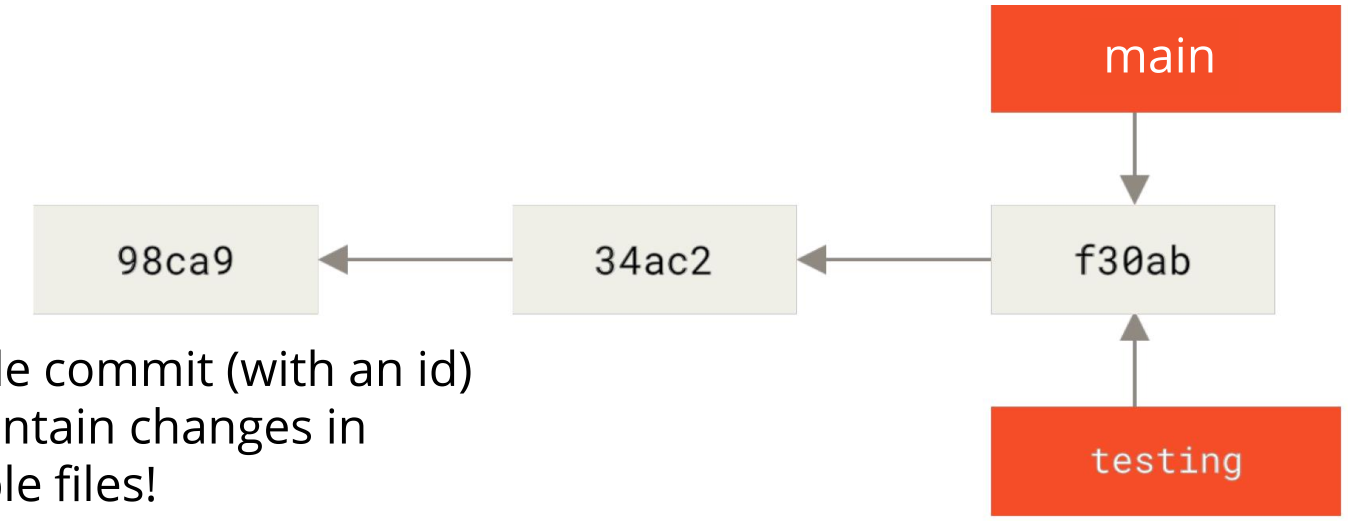
- The previous *"pole"* visualization only shows the status of a single file.

- We will use a graph visualization instead.



Commit Graph

This is a branch not a commit.
It refers to a specific commit.



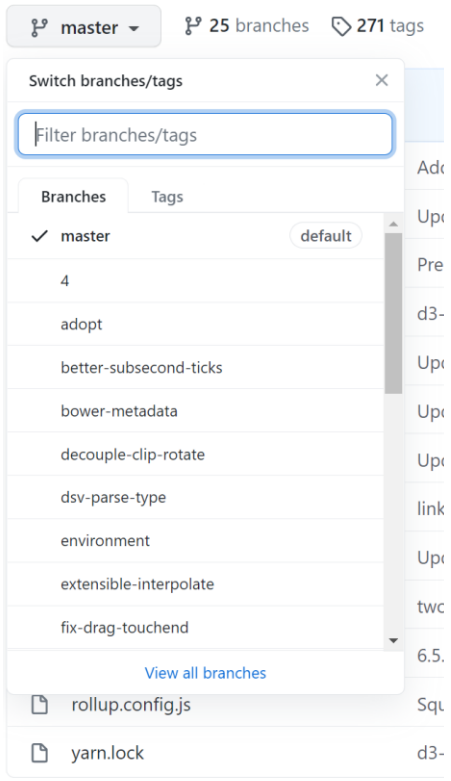
A single commit (with an id) can contain changes in multiple files!

Branching

- **Branching** means you diverge from the main line of development and continue to do work without messing with that main line.

- The default branch: *main*
  - The default was *master* but changed a few years ago.
  - Many old references still use the term *master*.
  - *Master* branch = *main* branch

- Branch for a new version
- Branch for a new feature
- Branch for an urgent fix



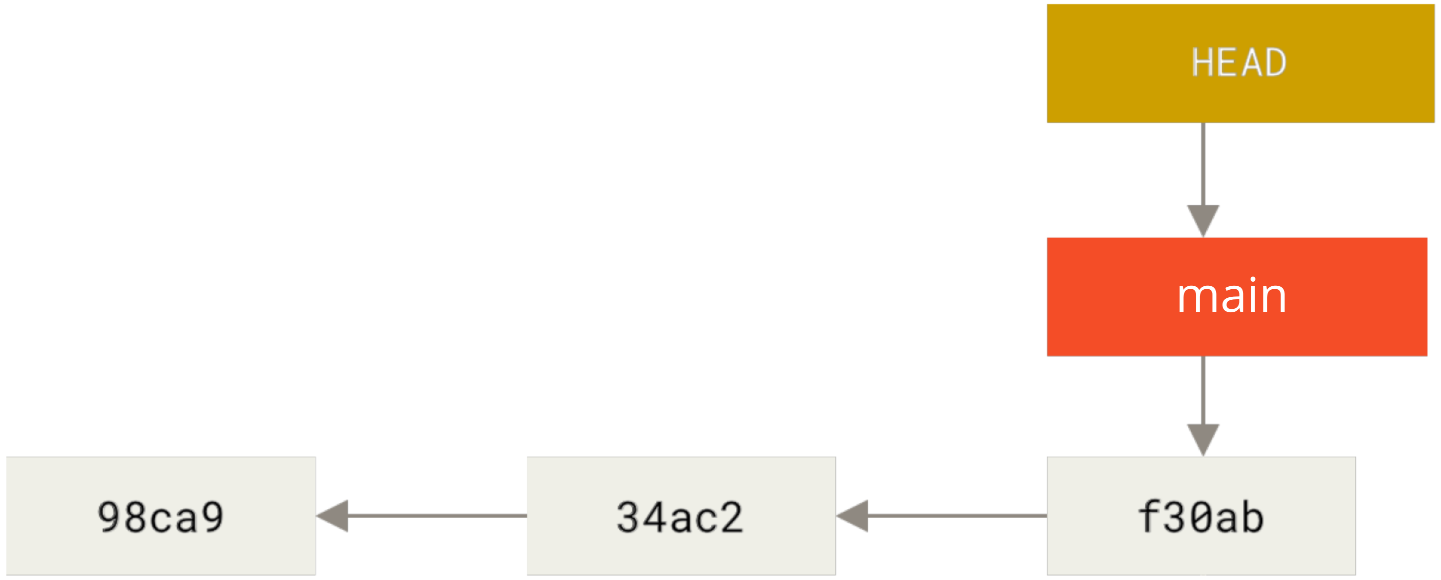
Branching Scenario

- You had deployed a web service (v1.0).
- You were developing v1.1.
- A bug is found in v1.0.
- You need to fix the source code of v1.0 without losing your work on v1.1.

- How would you do this?

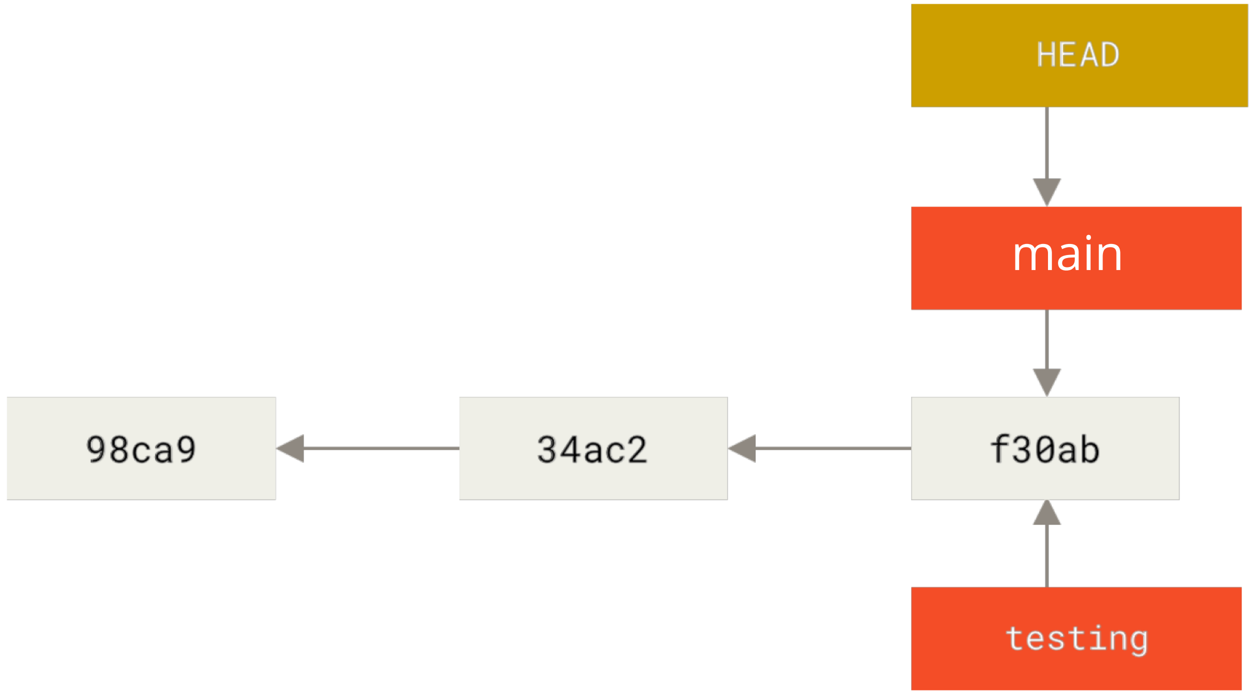
HEAD

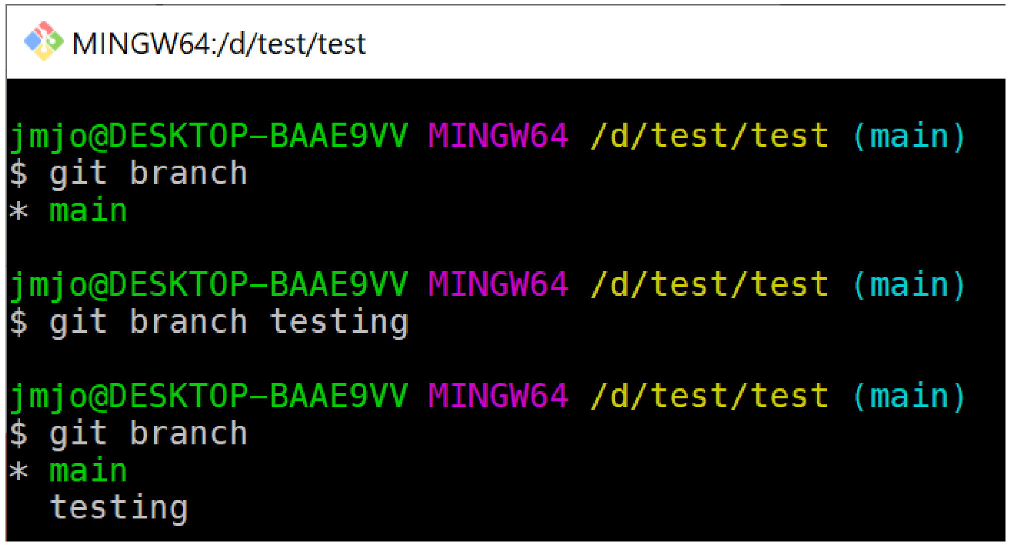
- HEAD is a special pointer to the last commit that you are on.
    - It can point to a branch, a tag, or a commit.
    - In this class, however, you can assume that it only refers to a branch.


$$\begin{tikzcd} & & \text{HEAD} \arrow[d] \\ & & \text{main} \arrow[d] \\ \text{98ca9} & \text{34ac2} \arrow[l] & \text{f30ab} \arrow[l] \end{tikzcd}$$

Creating a Branch

- `git branch <name>` creates a new branch <name> based on HEAD.
- `git branch` lists all branches and the branch that you are now on.

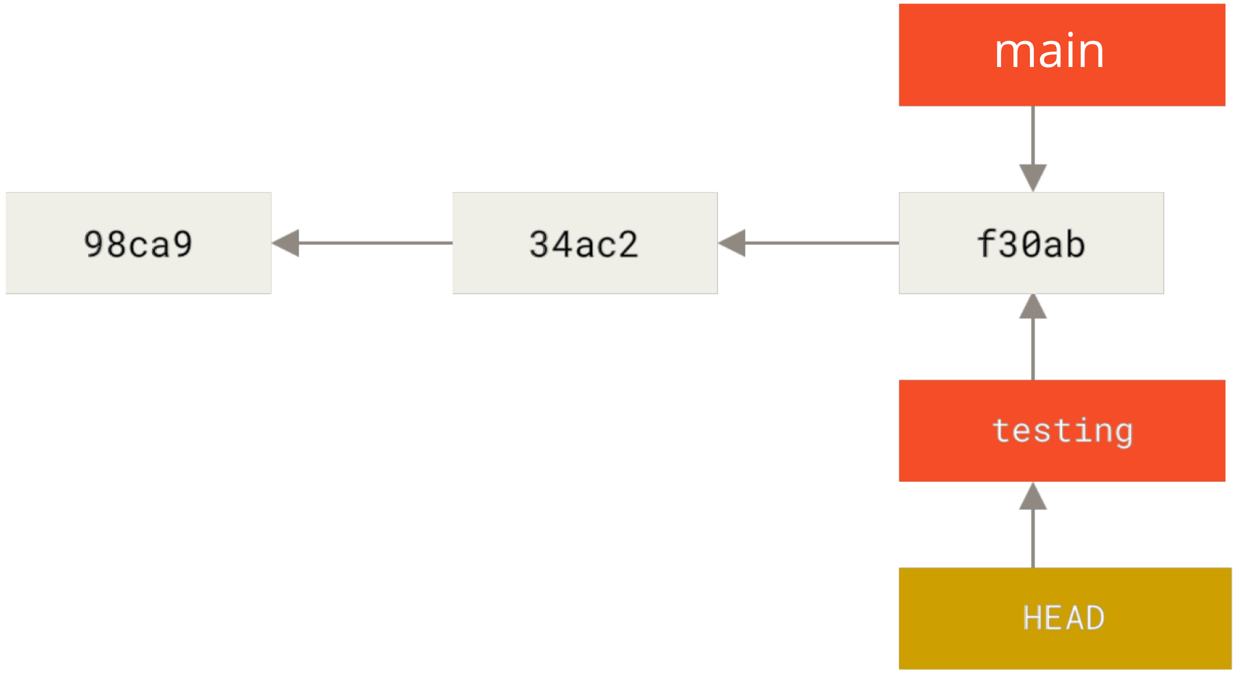


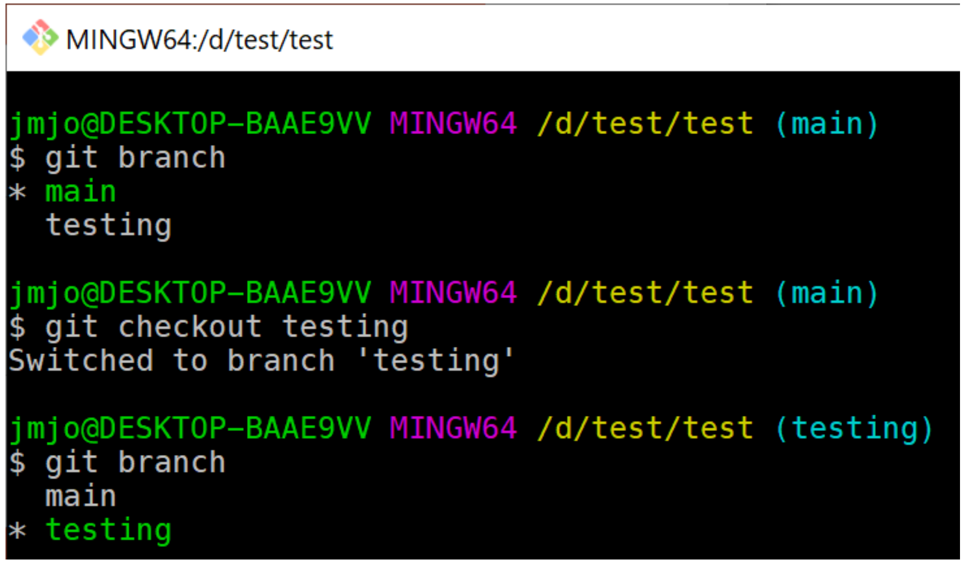

MINGW64:/d/test/test

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git branch
* main

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git branch testing

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git branch
* main
  testing
```

Switching Branches

- `git branch <name>` does not switch to the new branch.
- Run `git checkout <name>` to switch

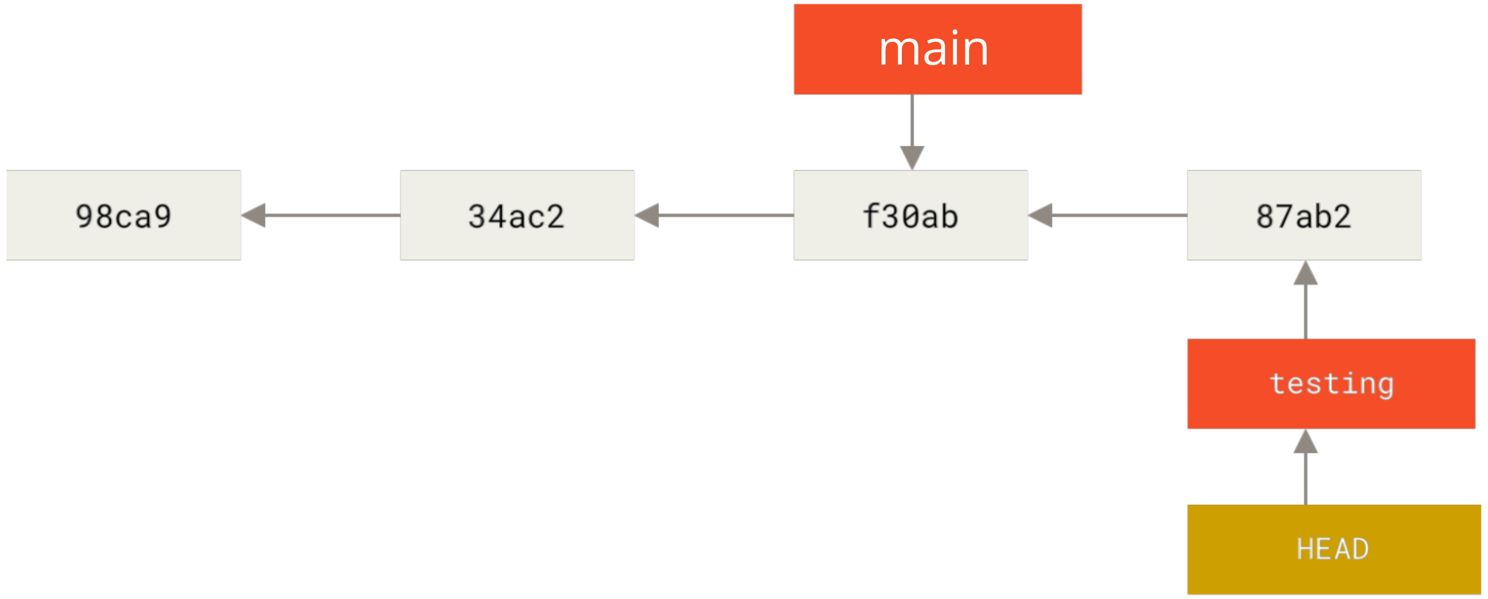




```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git branch
* main
  testing

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git checkout testing
Switched to branch 'testing'

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (testing)
$ git branch
  main
* testing
```

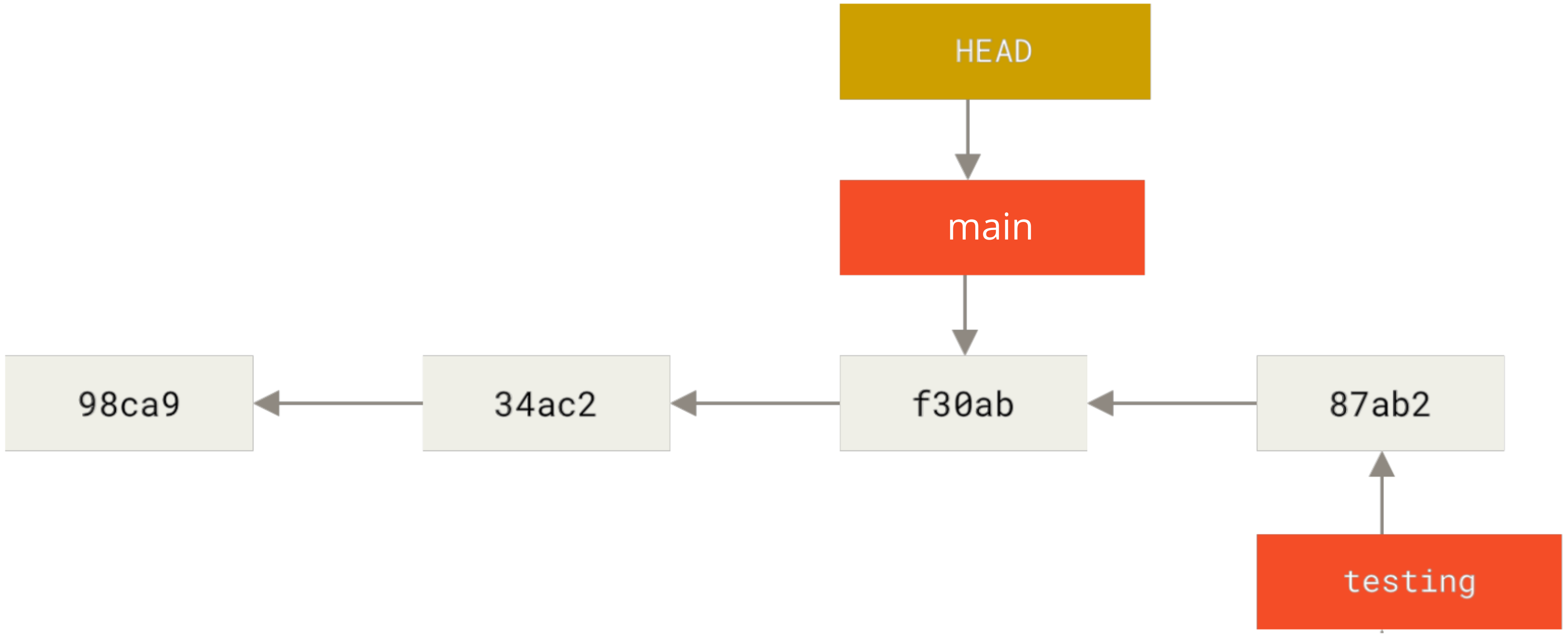
Commit after Switching

- If you do a commit, the commit is made on the branch that you are now on (the HEAD branch), moving forward the branch by one commit.



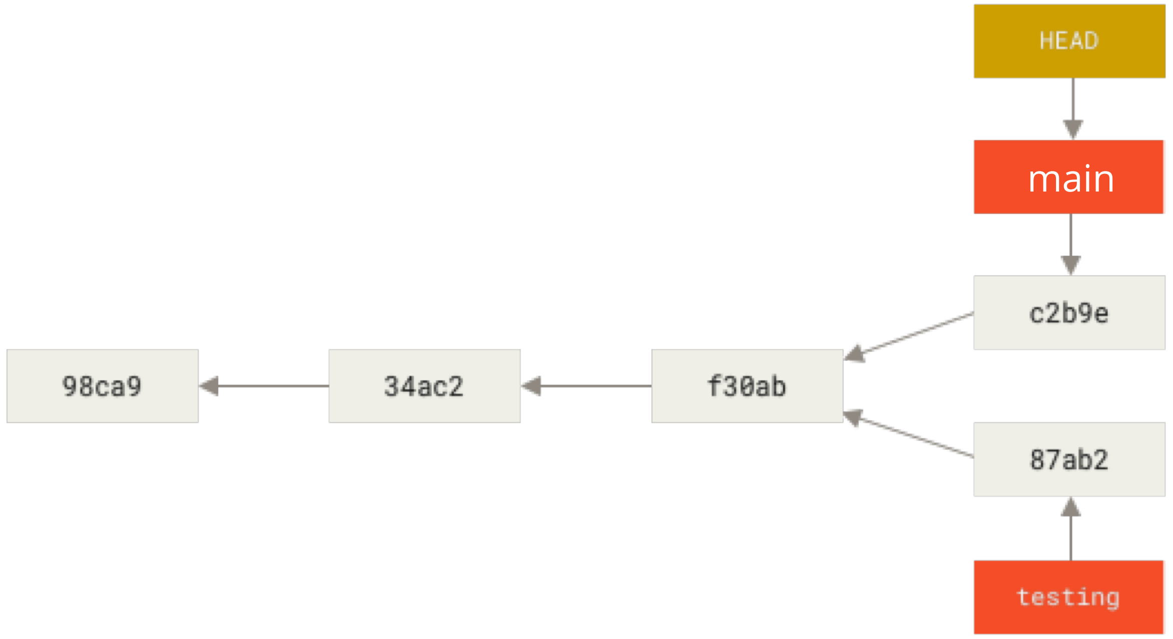
Going Back to Main

- You can go back to the main branch by running `git checkout main`.

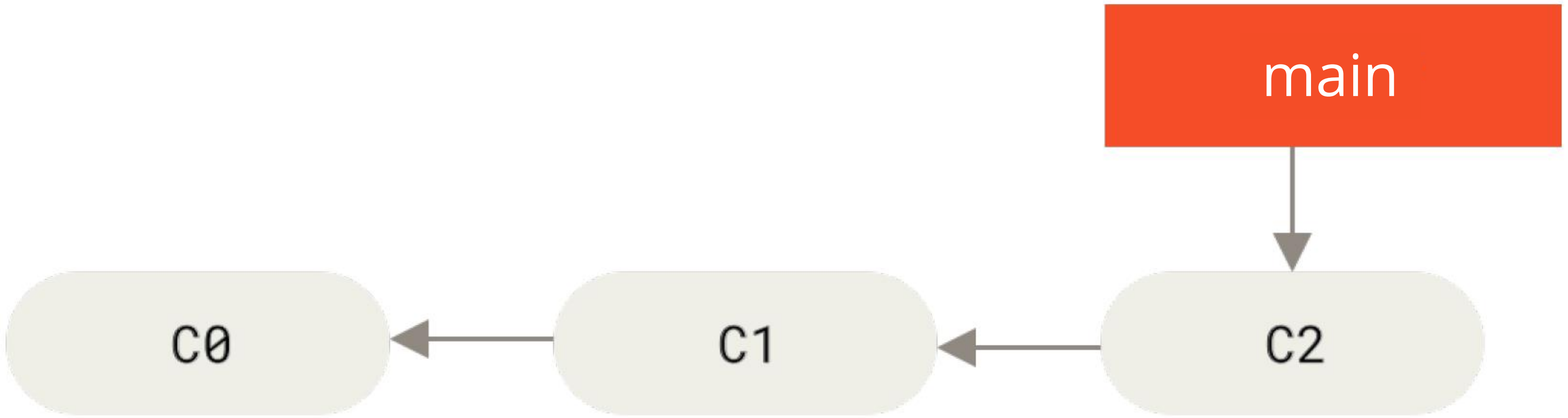


Doing a Commit on Main

- The HEAD branch is *main* now. So, if you do a commit, the commit is made on the main branch.
- Project history becomes diverged (this is very common).



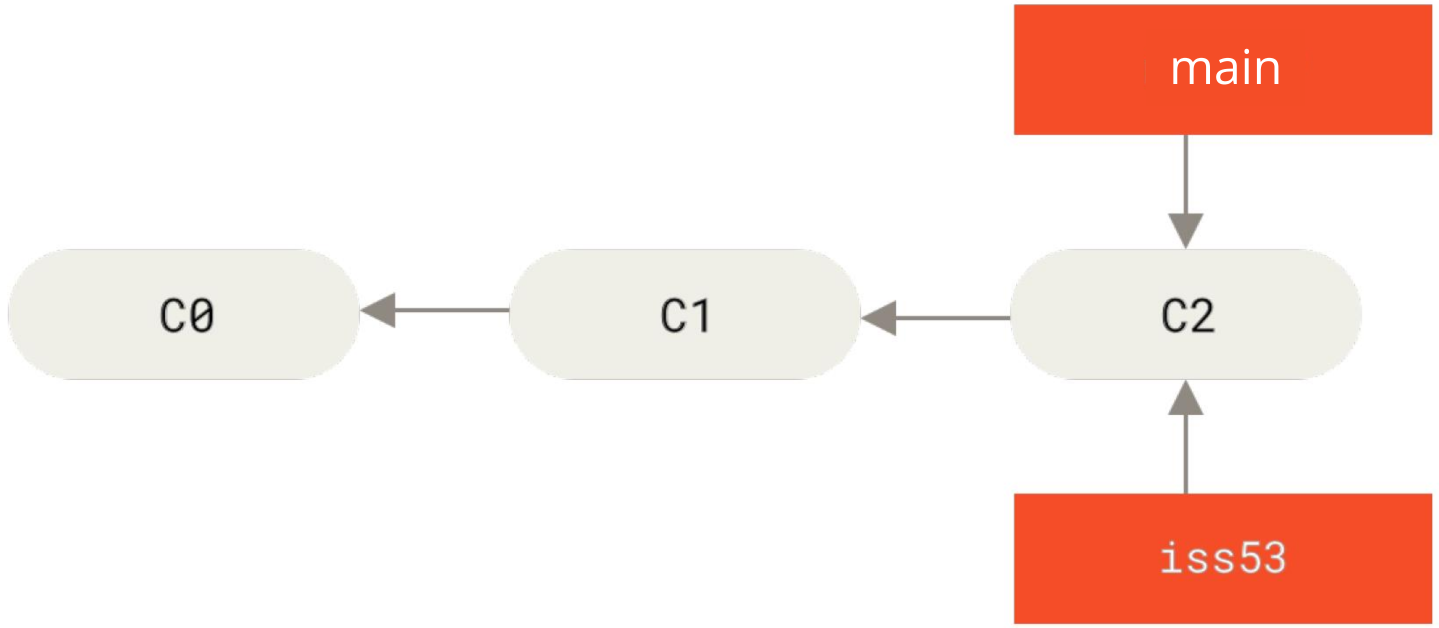
Branching Scenario



Branching Scenario

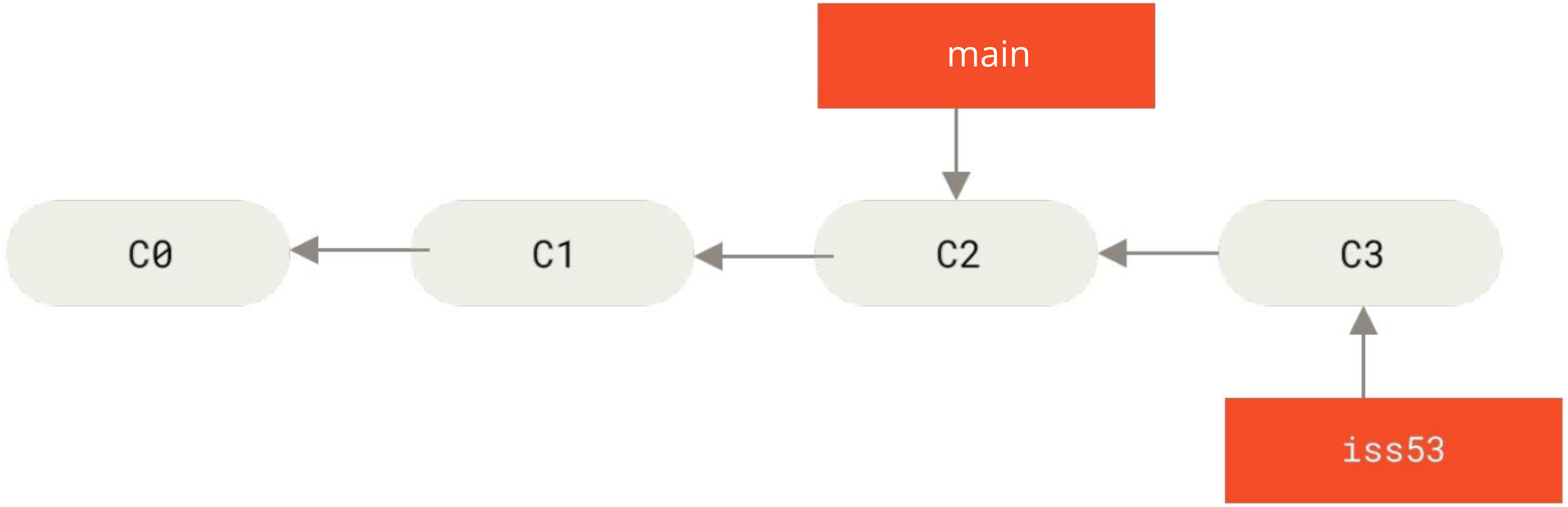
- git checkout –b iss53

- git branch iss53
- git checkout iss53



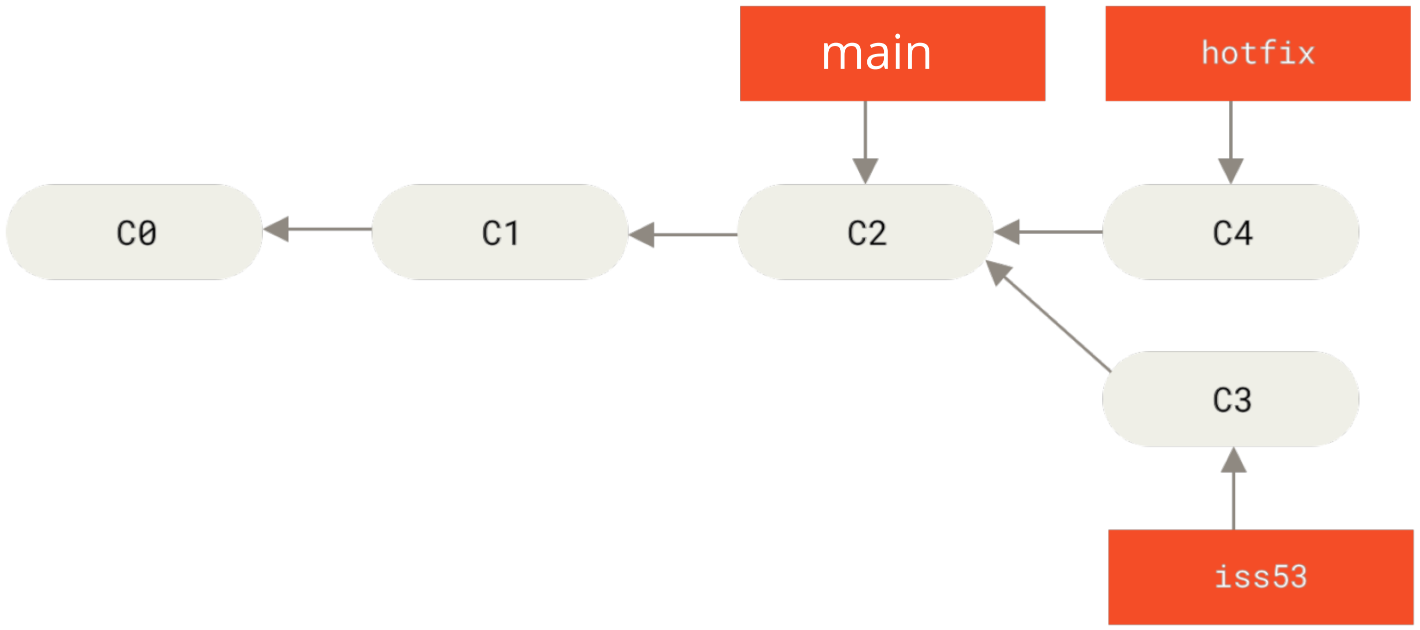
Branching Scenario

- `git commit`



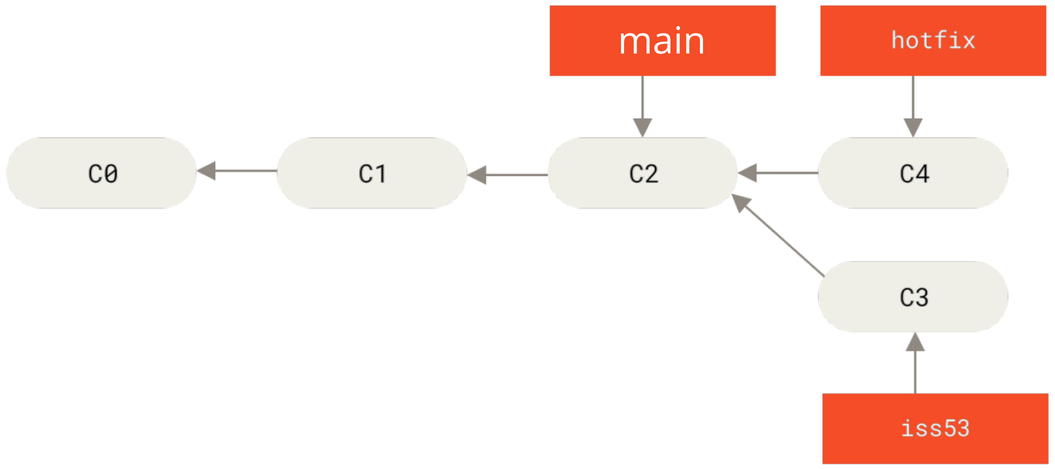
Branching Scenario

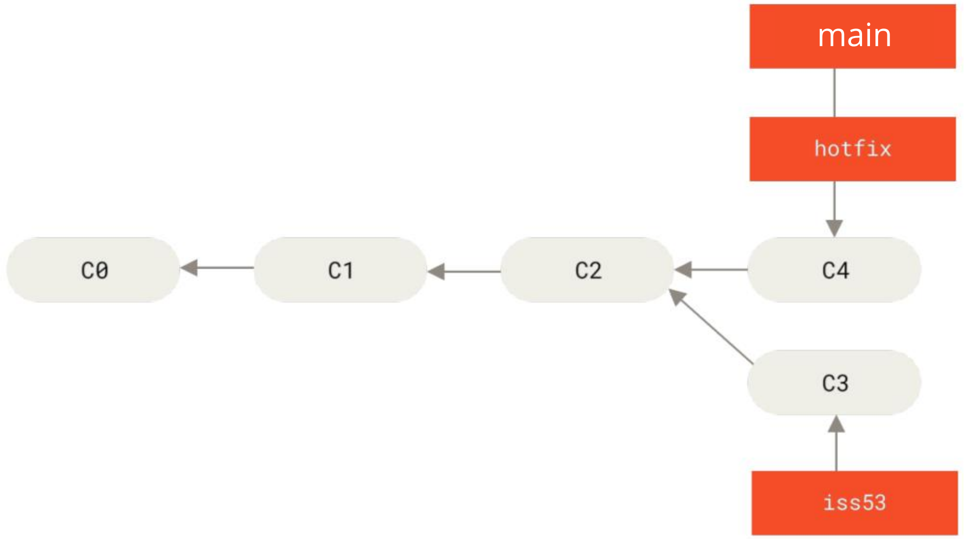
- `git checkout main`
- `git checkout –b hotfix`
- `Edit the code`
- `git commit`



Merging Branches

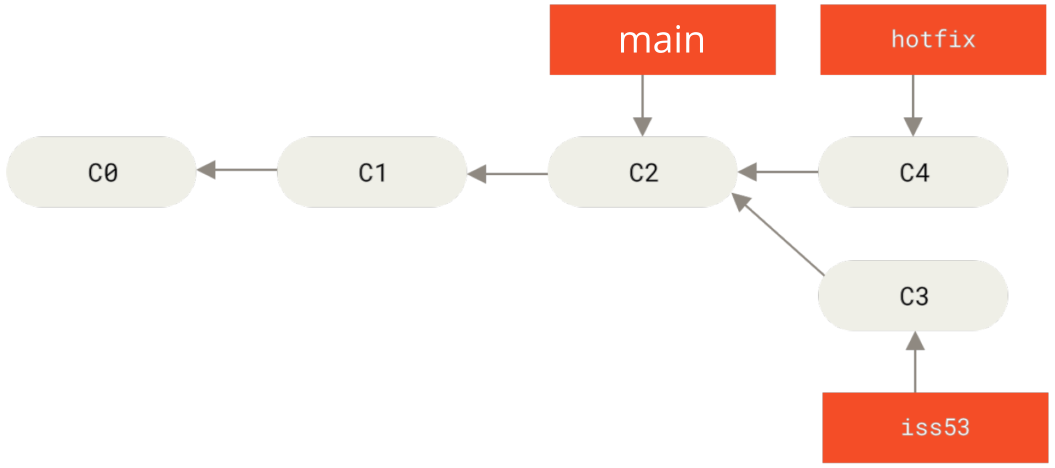
- The bug is fixed, and you now want to merge the *hotfix* branch back into your *main* branch to deploy to production.
- `git checkout main`
- `git merge hotfix`
- Fast-forward merge in this case (빨리감기).

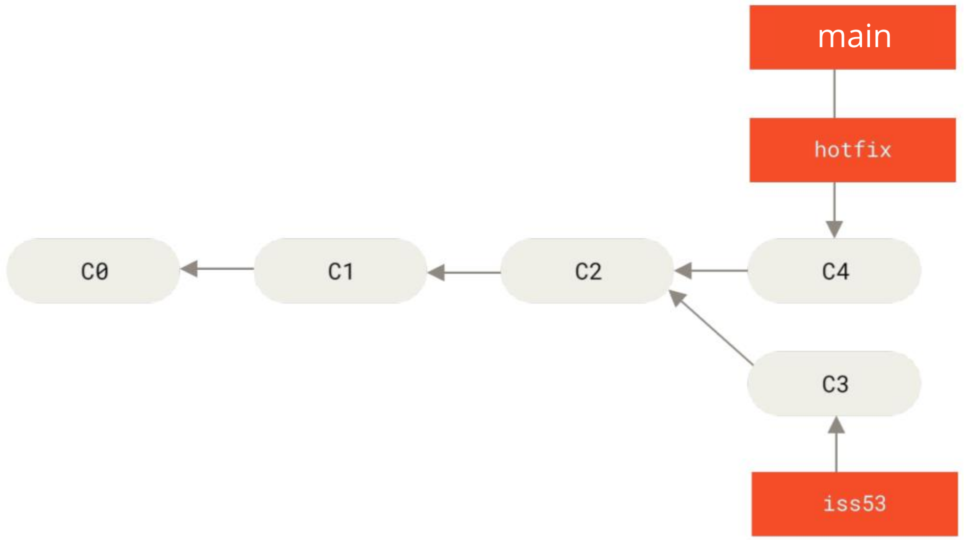




Merging Branches

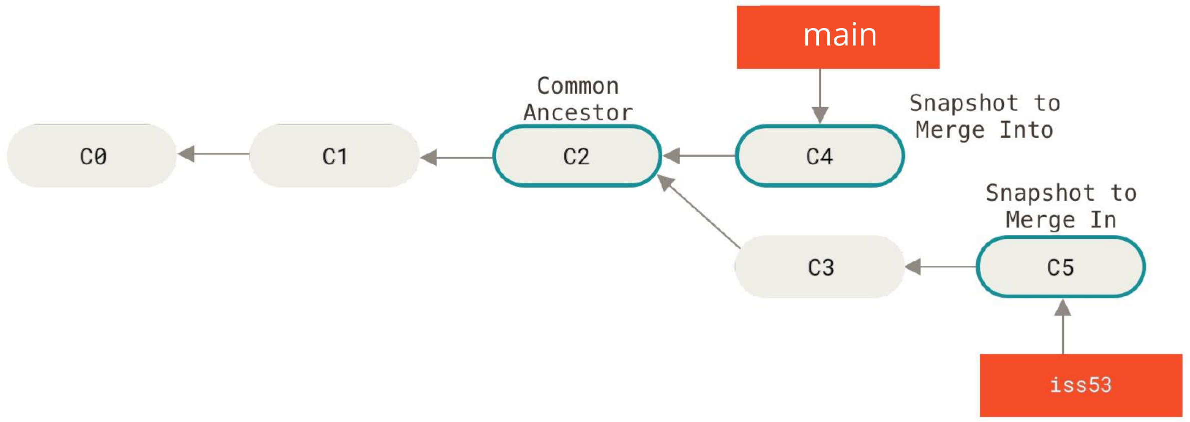
- The order is important:
  - `git checkout` <the branch you want to update, usually the main branch>
  - `git merge` <the branch you want to merge to, usually the newer branch>
- After merging, go back to *iss53* and continue working…
  - `git checkout iss53`

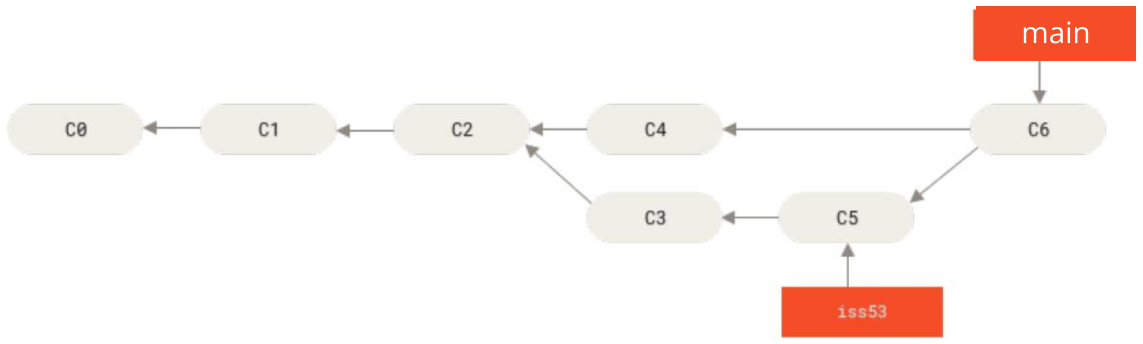




Merging Branches

- Your work on *iss53* is done, so you want to merge it to the *main* branch.
- `git checkout main`
- `git merge iss53`
- How would this be if we checkout iss53 and then merge main?





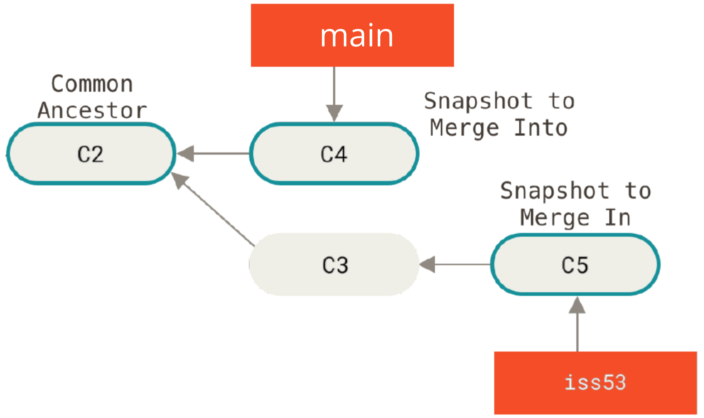
Merge Conflicts

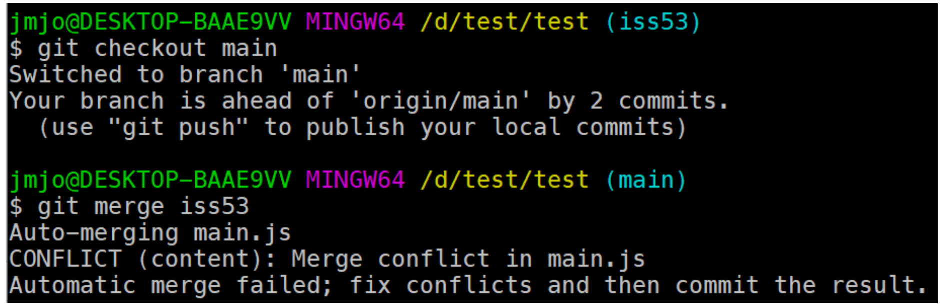
- **A merge conflict** happens during merging if you changed the same part of the same file differently in the two branches you're merging.

main.js in C2
let a;

main.js in main (C4)
let a = 1;

main.js in iss53 (C5)
let a, b;

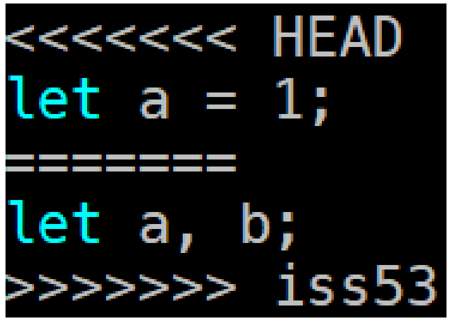




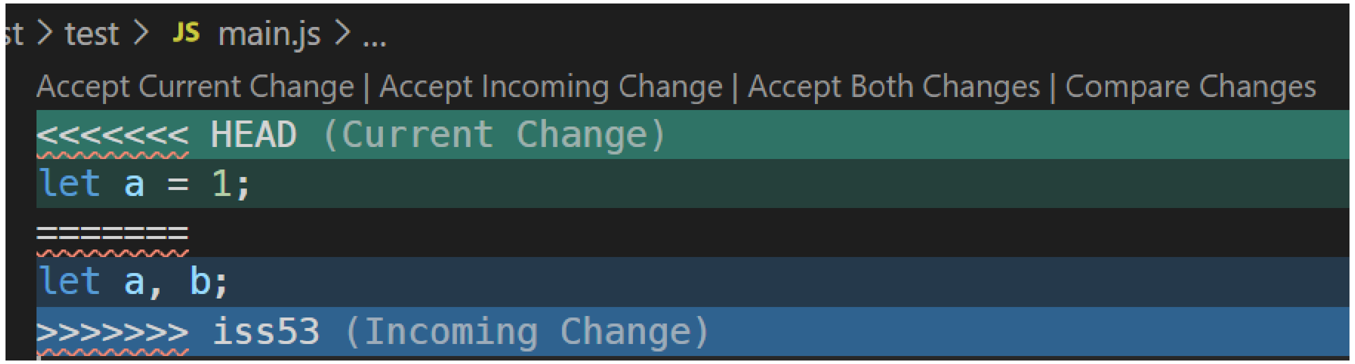
```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (iss53)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git merge iss53
Auto-merging main.js
CONFLICT (content): Merge conflict in main.js
Automatic merge failed; fix conflicts and then commit the result.
```

Merge Conflicts

- If merge conflicts happen, Git will **append** conflict messages to your code that caused the conflicts.
- You need to make a **merge commit** that removes these messages and resolves the conflicts.
- If you leave some messages in your code, the code will not be compiled.



```
<<<<<<< HEAD
let a = 1;
=======
let a, b;
>>>>>>> iss53
```

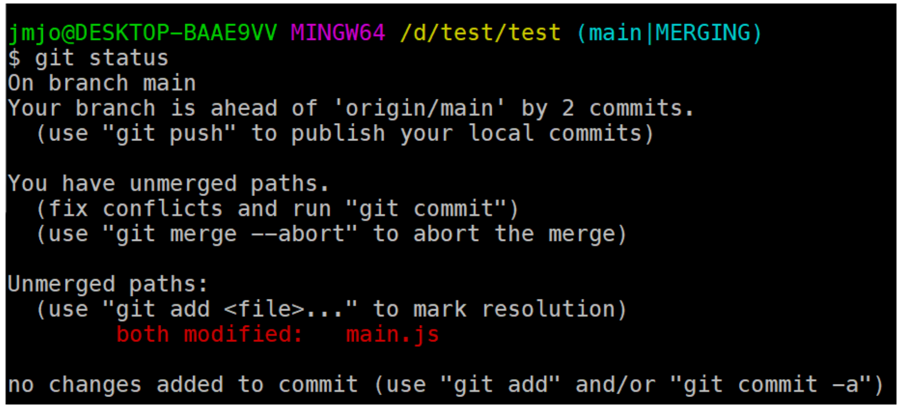


st > test > JS main.js > ...
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

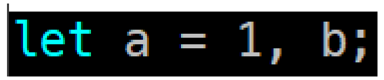
```
<<<<<<< HEAD (Current Change)
let a = 1;
=======
let a, b;
>>>>>>> iss53 (Incoming Change)
```

Merge Conflicts

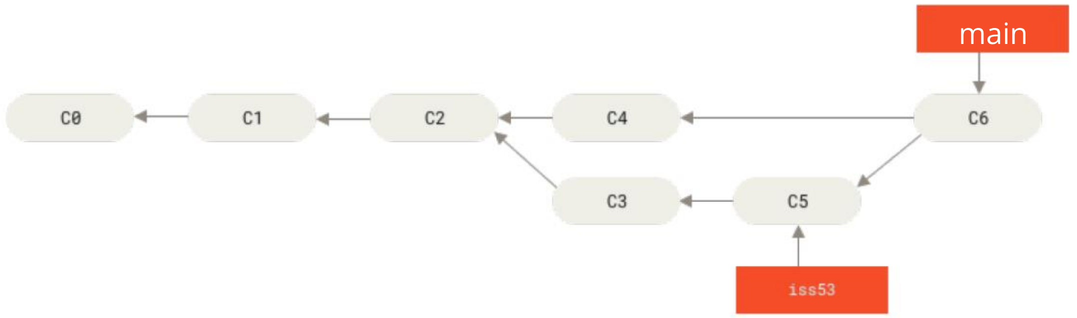
- Resolve the conflicts.
- `git add .`
- `git commit`



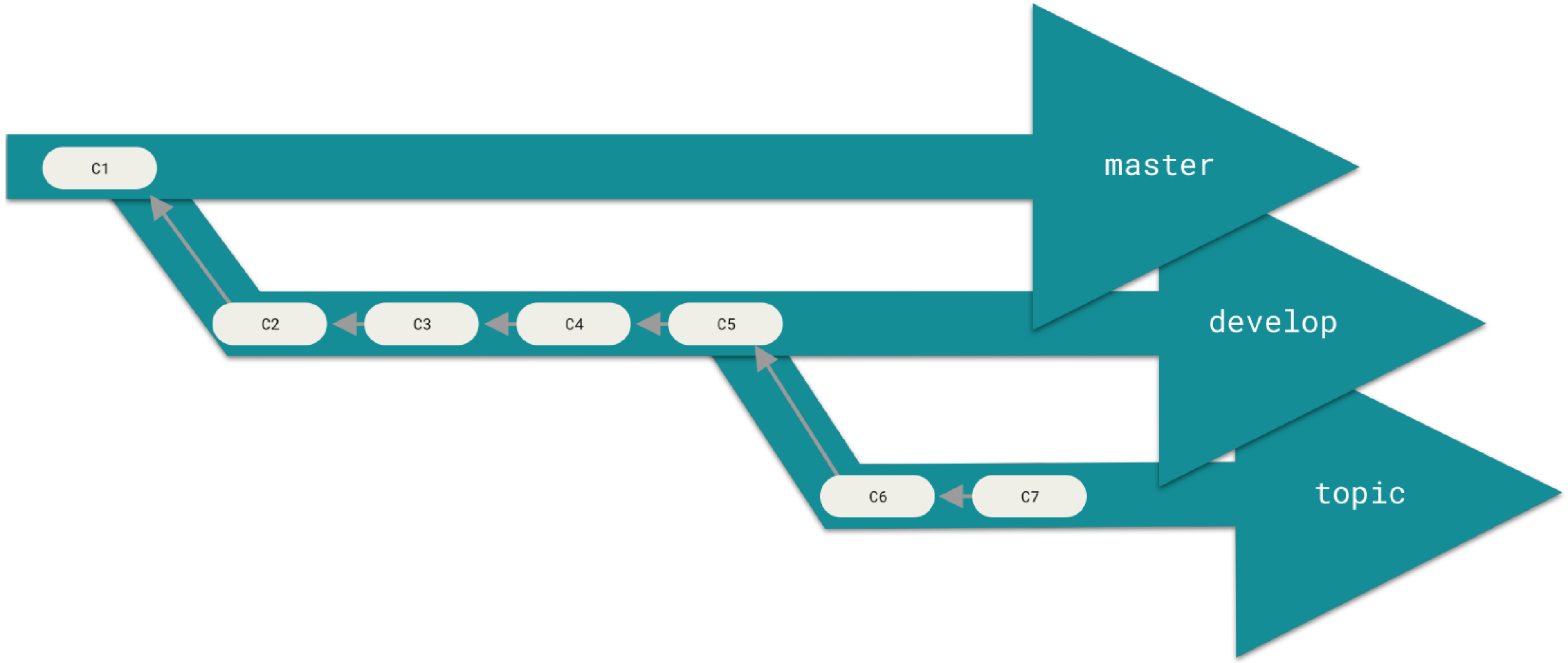
```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main|MERGING)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   main.js

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

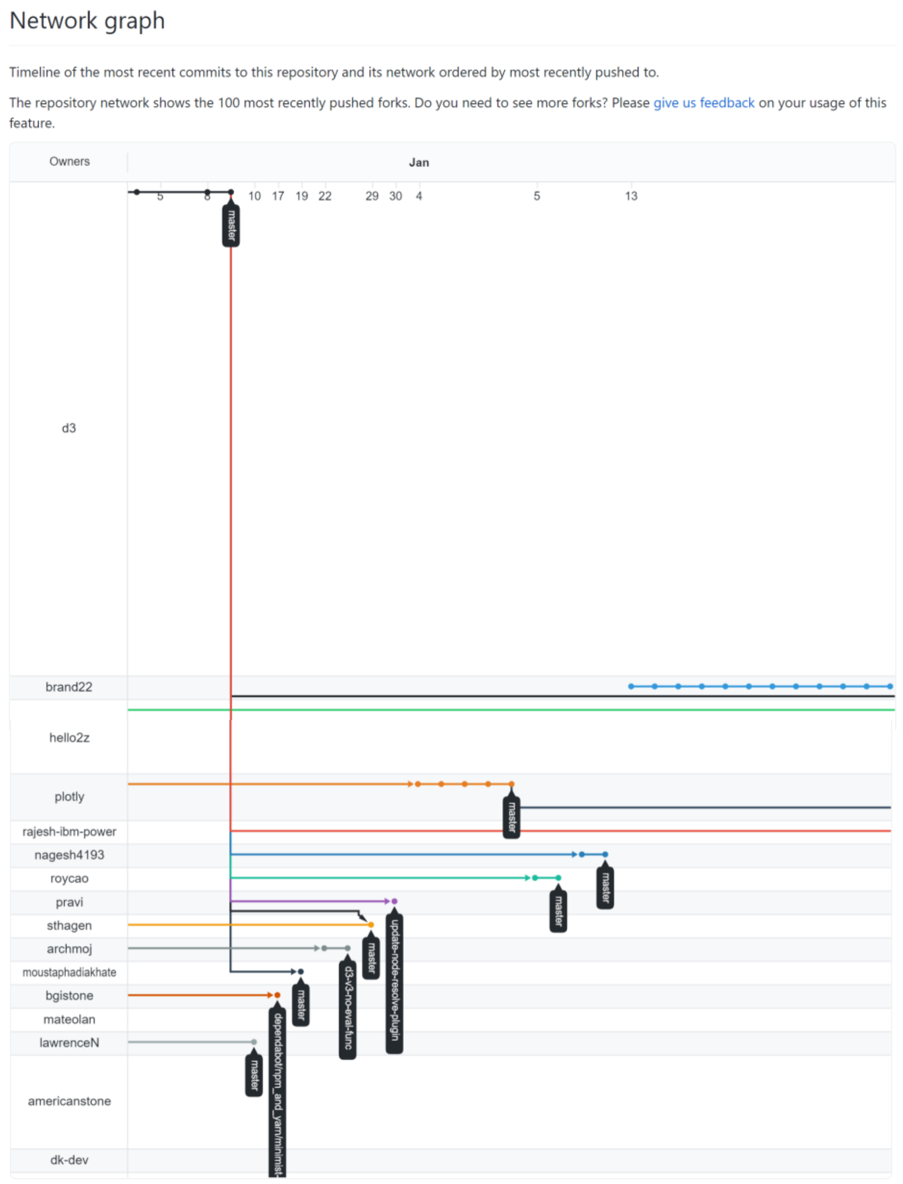


```
let a = 1, b;
```



Branching Workflow





Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

The repository network shows the 100 most recently pushed forks. Do you need to see more forks? Please give us feedback on your usage of this feature.

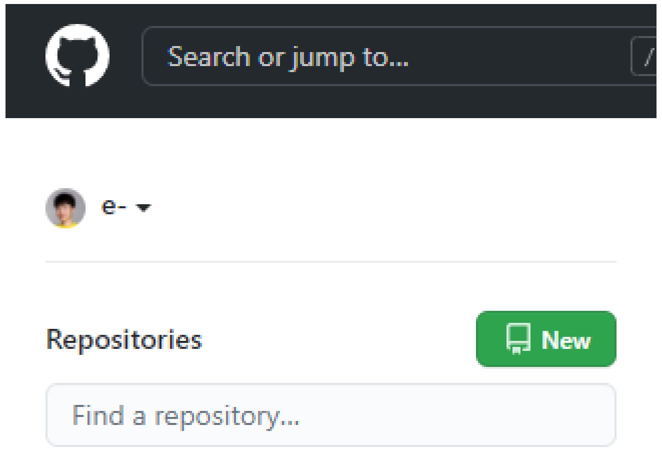
Working with Remotes

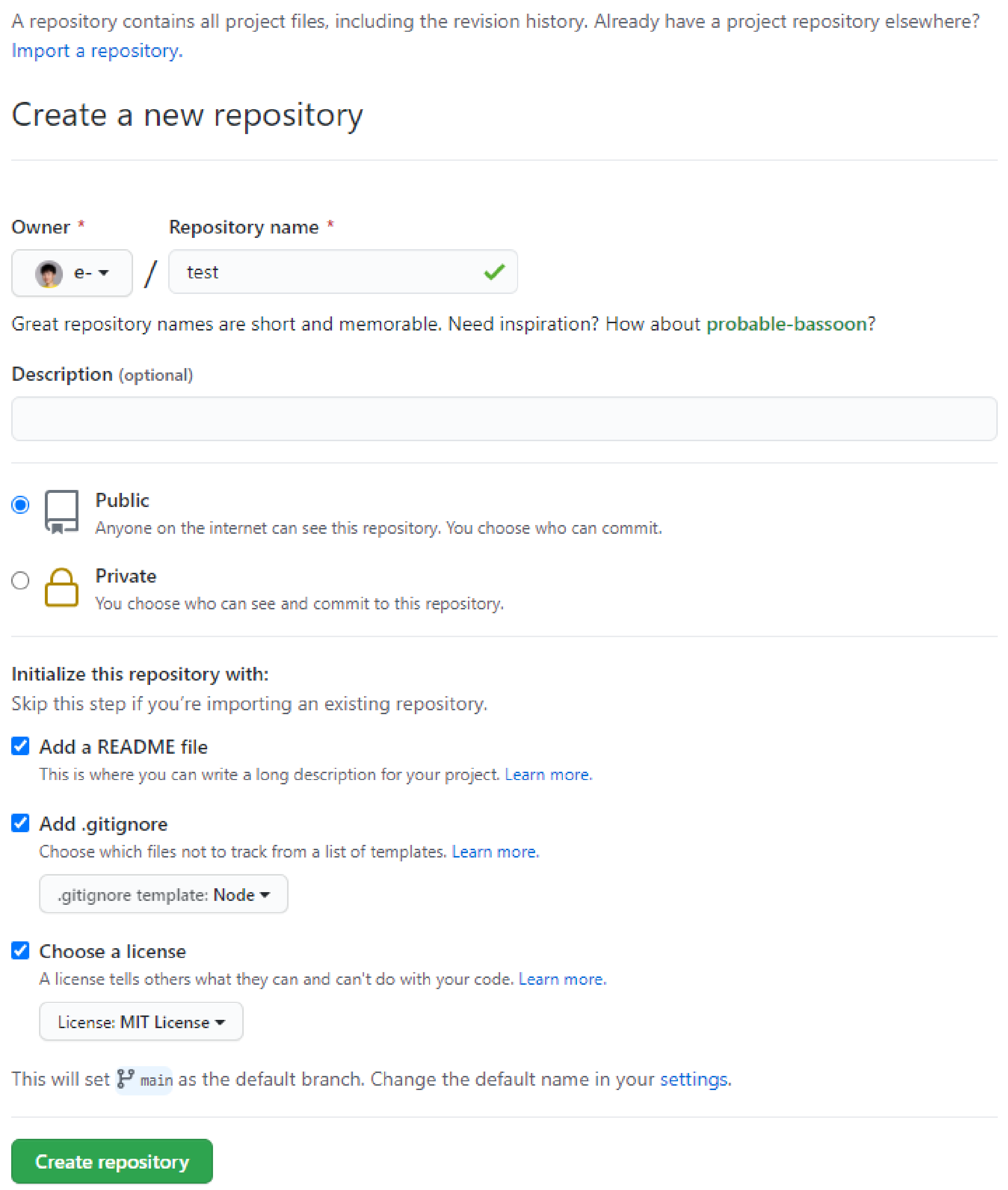
- So far, your commits were stored only in your local machine.
  - We didn’t even use GitHub.

- **Remote repositories** are versions of your project that are hosted on the Internet or network somewhere.

Creating a Repository on GitHub

- You need a right credential that can authenticate your identity.
  - We will work on this in the lab session.




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

Create a new repository

Owner * / Repository name *

e- / test ✓

Great repository names are short and memorable. Need inspiration? How about probable-bassoon?

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.

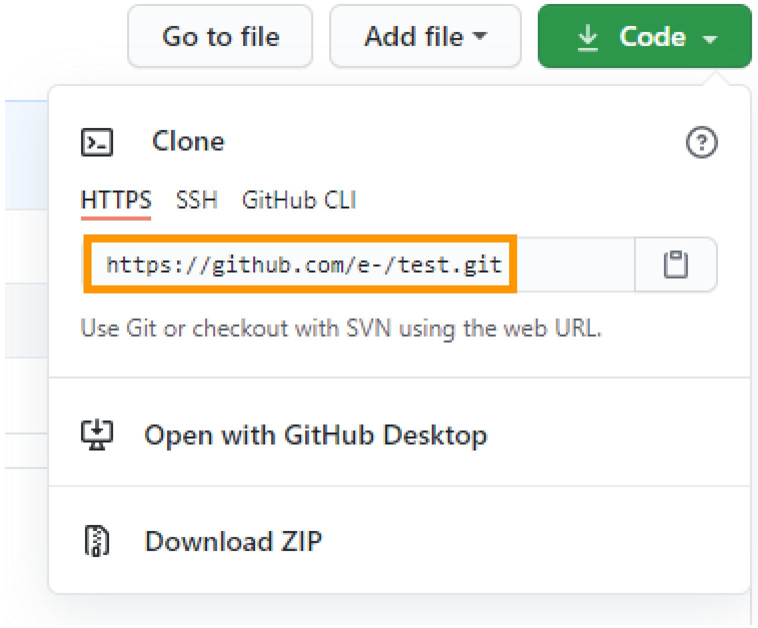
[✓] Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: Node

[✓] Choose a license
A license tells others what they can and can't do with your code. Learn more.

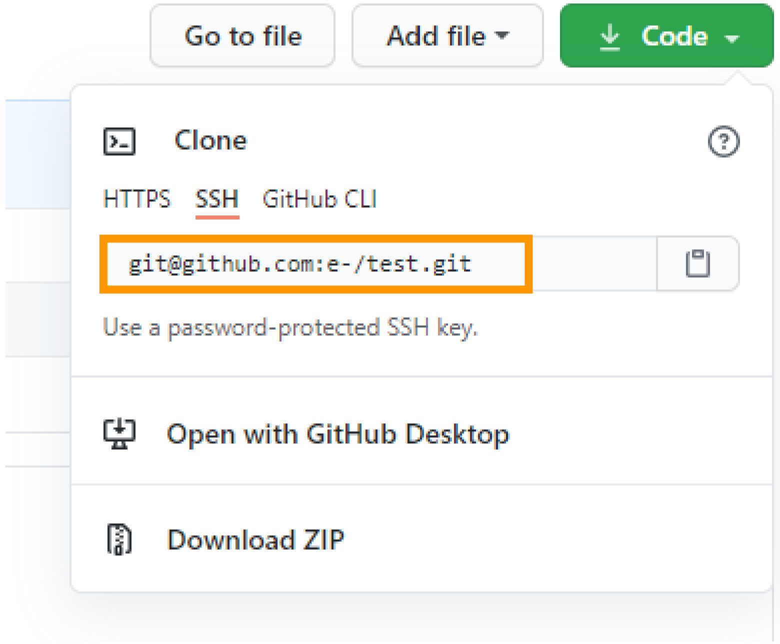
License: MIT License

This will set `main` as the default branch. Change the default name in your settings.


Go to file
Add file
Code
Clone
HTTPS SSH GitHub CLI

```
https://github.com/e-/test.git
```

Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP


Go to file
Add file
Code
Clone
HTTPS SSH GitHub CLI

```
git@github.com:e-/test.git
```

Use a password-protected SSH key.
Open with GitHub Desktop
Download ZIP

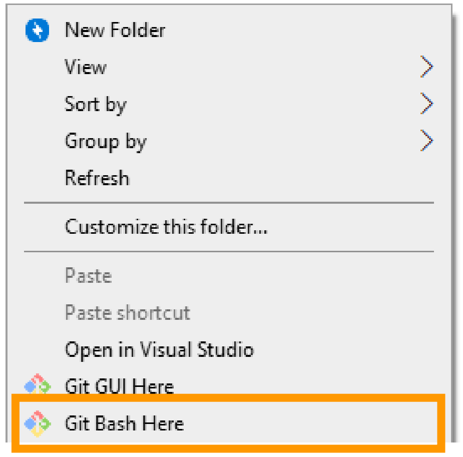
Cloning a Repository

- There are two urls for your repos:
  - https://github.com/e-/test.git
  - git@github.com:e-/test.git
- The https one uses temporary authentication.
  - It will ask your GitHub username and password every time you push your commits to the remote.
  - Very annoying
- Let’s use the “git” one.

Cloning a Repository

1. Go to a directory where you want to clone a repo.
   - In my case, it is “D:\”.
   - Don’t create a subdirectory with the project name like “D:\test”.

2. Right-click and open a Git Bash.



3. `git clone <your git url>`

```
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

4. Error! Authentication Failed.

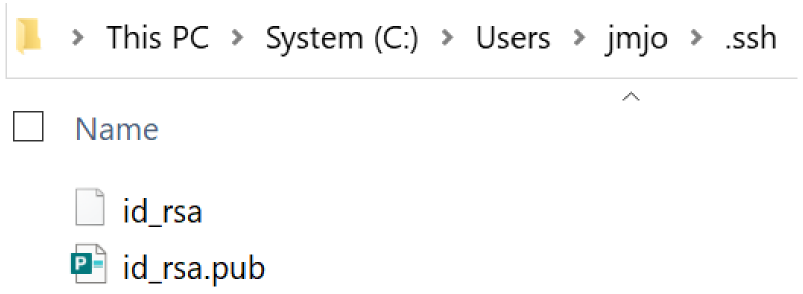
Git Authentication

- GitHub does not know who you are.

1. Issue public and private keys (credentials).
   - Usually, stored in ~/.ssh

2. Register the public key to GitHub.
   - Keep the private key as a secret. Never share the private key.

3. The `git` command will automatically use the keys under ~/.ssh if they exist.


This PC › System (C:) › Users › jmjo › .ssh

Name

id_rsa

id_rsa.pub

Issue the Two Keys

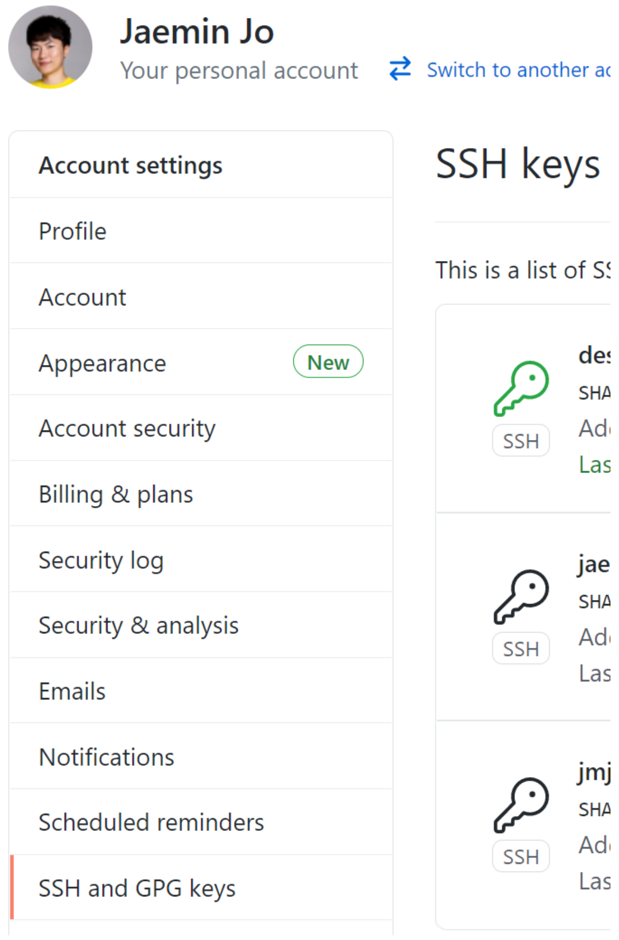
- In short: run “ssh-keygen” on your console.
- The command comes with the Git installation (Git for Windows).

- https://git-scm.com/book/en/v2/Git-on-the-Server-Generating-Your-SSH-Public-Key

Register the Keys

1. Go to SSH and GPG keys tab in Settings.
2. Enter your password again
3. Click on “New SSH Key”
4. Copy the content of **“id_rsa.pub”** to the form.
   - **NOT** “id_rsa”. This is your private key.
   - The `ssh-keygen` command will print out the public key on the console after the keys are generated.

`5. git clone`



Jaemin Jo
Your personal account
⇄ Switch to another ac

Account settings
Profile
Account
Appearance New
Account security
Billing & plans
Security log
Security & analysis
Emails
Notifications
Scheduled reminders
SSH and GPG keys

SSH keys

This is a list of S

des
SHA
Ad
Las
SSH

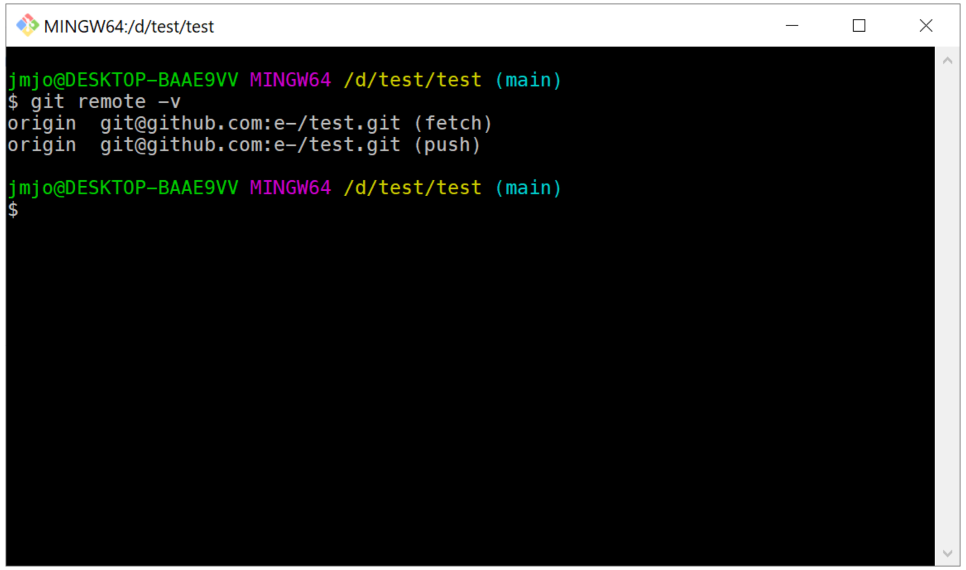
jae
SHA
Ad
Las
SSH

jmj
SHA
Ad
Las
SSH

Showing the Remotes

- `git remote`
- `git remote –v`

- “origin” is the name of the default remote.


MINGW64:/d/test/test

```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git remote -v
origin  git@github.com:e-/test.git (fetch)
origin  git@github.com:e-/test.git (push)

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$
```

- “main” is the name of the default branch, we will learn this later.

Initializing a Git Repository

- `git clone <url>`
  - Initialize a repository by cloning a remote repository.
  - <url> automatically becomes the origin.

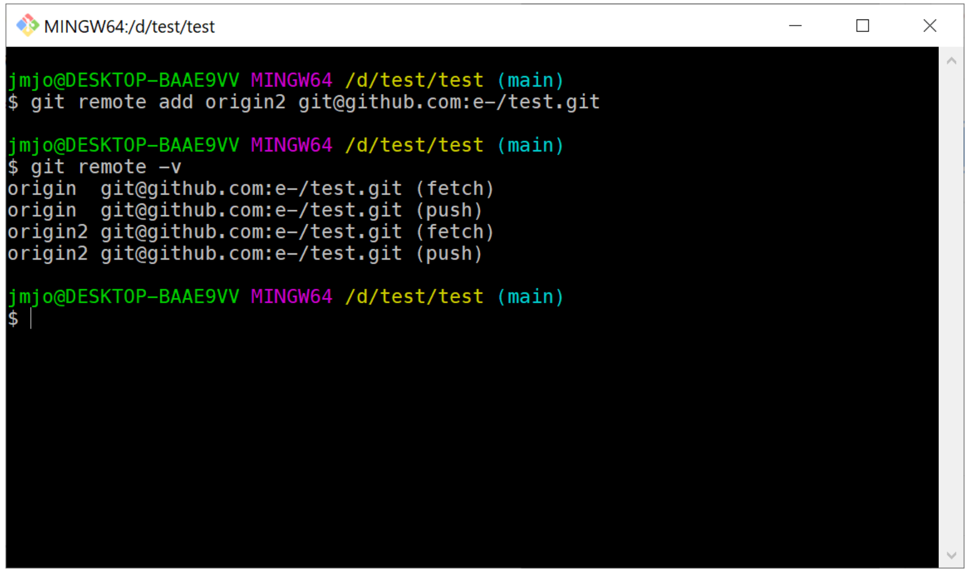
- `git init`
  - Initialize a repository from a local directory.
  - You can run all *local* Git commands, such as add, commit, …

  - You must add at least one remote to push your commits to a remote.

Working with Remotes

- `git remote add <shortname> <url>`
- `git remote rename <oldname> <newname>`
- `git remote rm <name>`

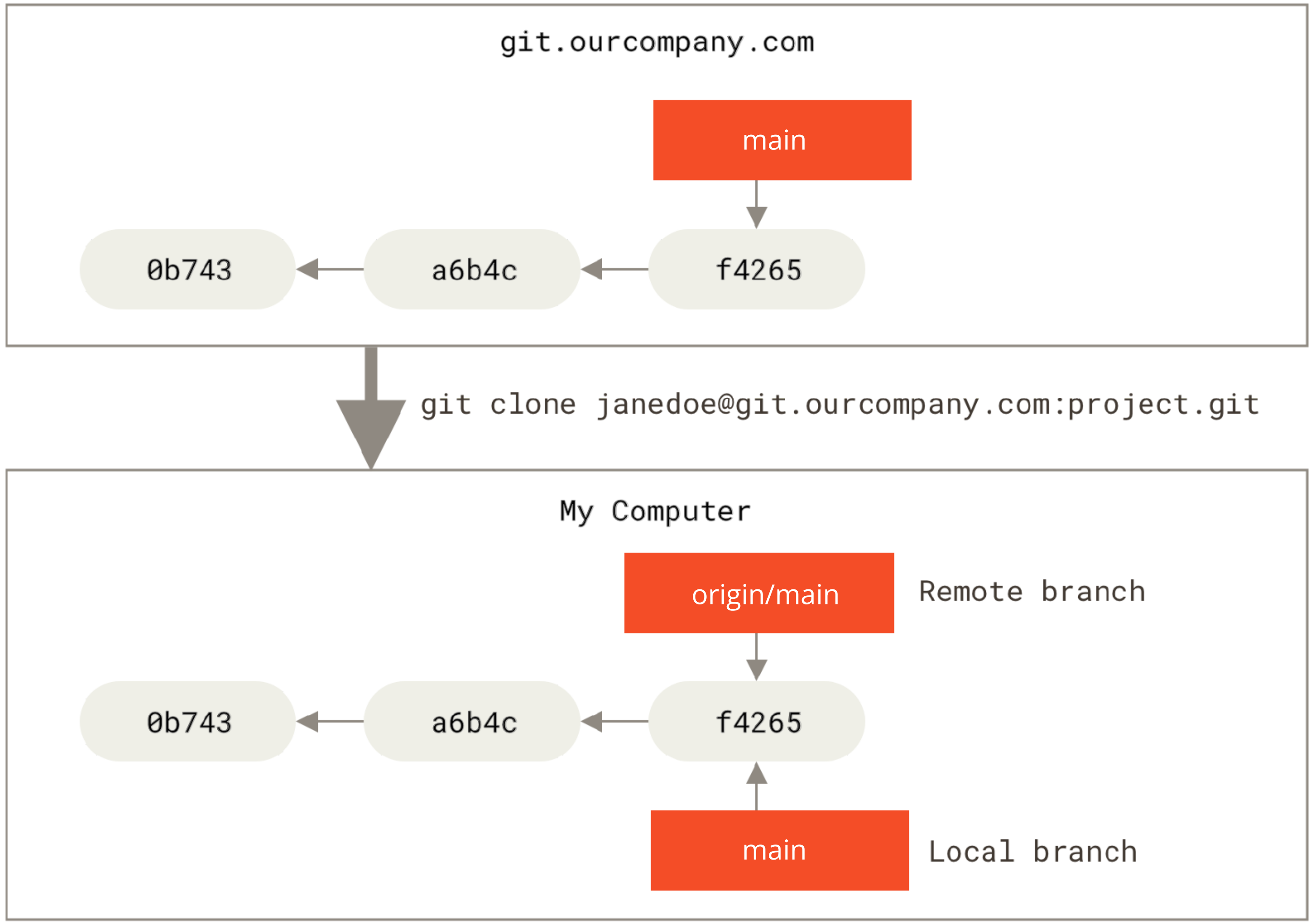
- You can register multiple remotes, but it would be more common to have only one remote, *origin*.



MINGW64:/d/test/test

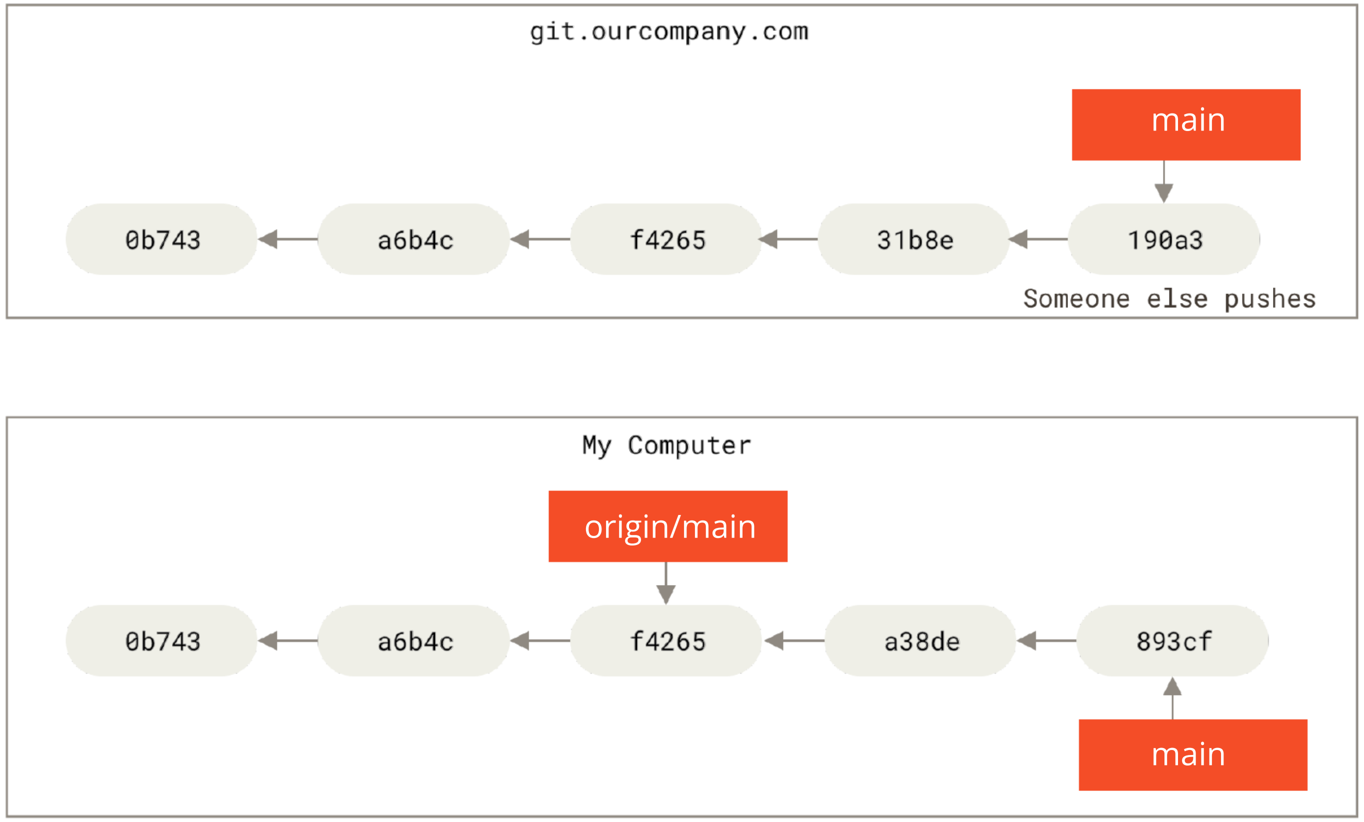
```
jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git remote add origin2 git@github.com:e-/test.git

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ git remote -v
origin  git@github.com:e-/test.git (fetch)
origin  git@github.com:e-/test.git (push)
origin2 git@github.com:e-/test.git (fetch)
origin2 git@github.com:e-/test.git (push)

jmjo@DESKTOP-BAAE9VV MINGW64 /d/test/test (main)
$ 
```

Remote Branches



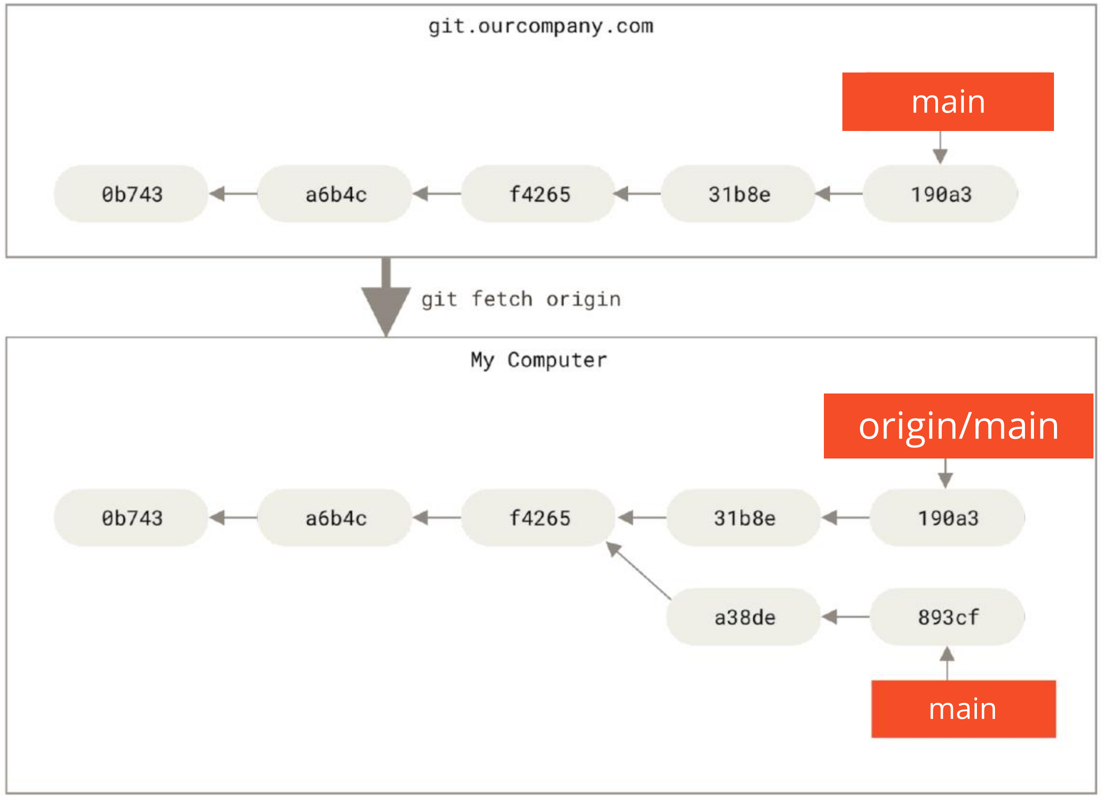
Remote Branches

- If others push to the remote, the local and remote work can diverge.



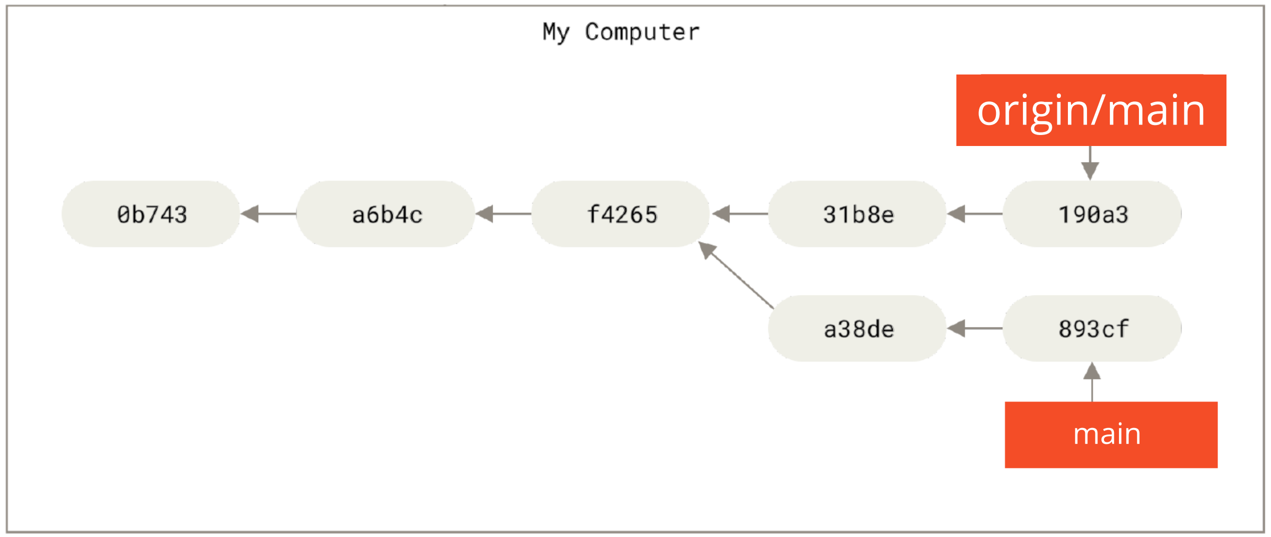
Updating the Remote Branches

- `git fetch <remote_name>` or just `git fetch` if you fetch the default remote, *origin*.



Updating the Remote Branches

- `git fetch` is a "safe" command; it doesn't integrate any of new data into your local branches.



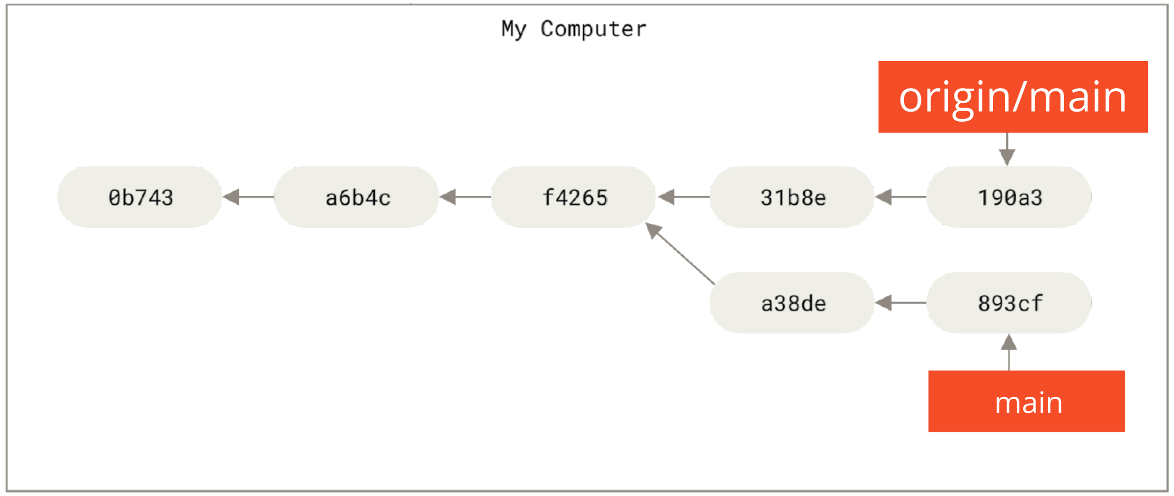
Pushing to the Remote

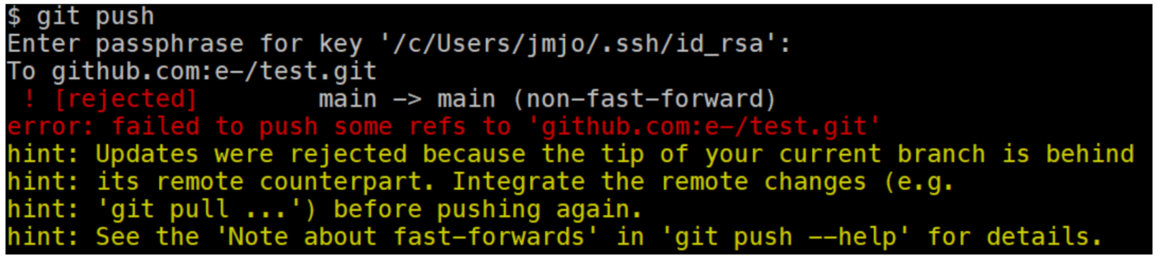
- `git push` pushes your current local branch to the remote.

- `git push` pushes the current branch to its remote (*origin* by default).
- `git push <remote_name>` if you want to push the current branch to <remote_name>
- `git push <remote_name> <branch_name>` if you want to push a specific branch to <remote_name>.

Pushing to the Remote

- Pushing to the remote can cause merge conflicts.
  - This is the same as the conflicts between two local branches.
  - Merge conflicts between a *local* version of a branch and a *remote* version of the branch

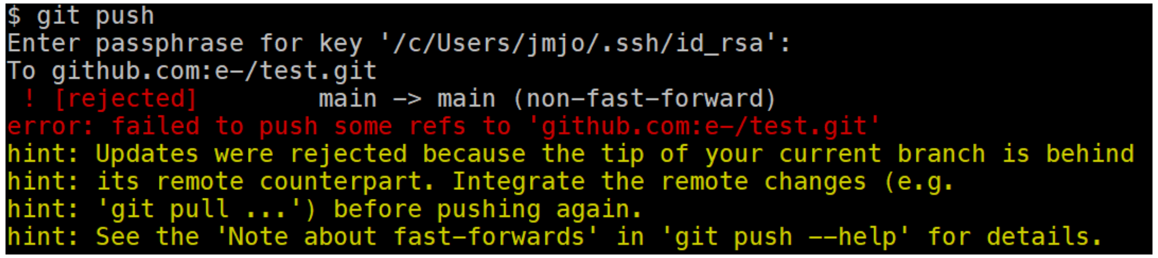




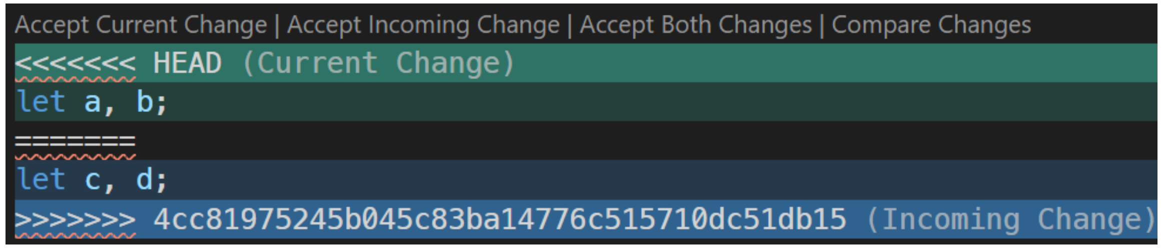
```
$ git push
Enter passphrase for key '/c/Users/jmjo/.ssh/id_rsa':
To github.com:e-/test.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'github.com:e-/test.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote 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.
```

Pulling the Changes

- Run `git pull` to integrate the remote changes.
- Resolve the conflicts.
- `git add .`
- `git commit`
- `git push`



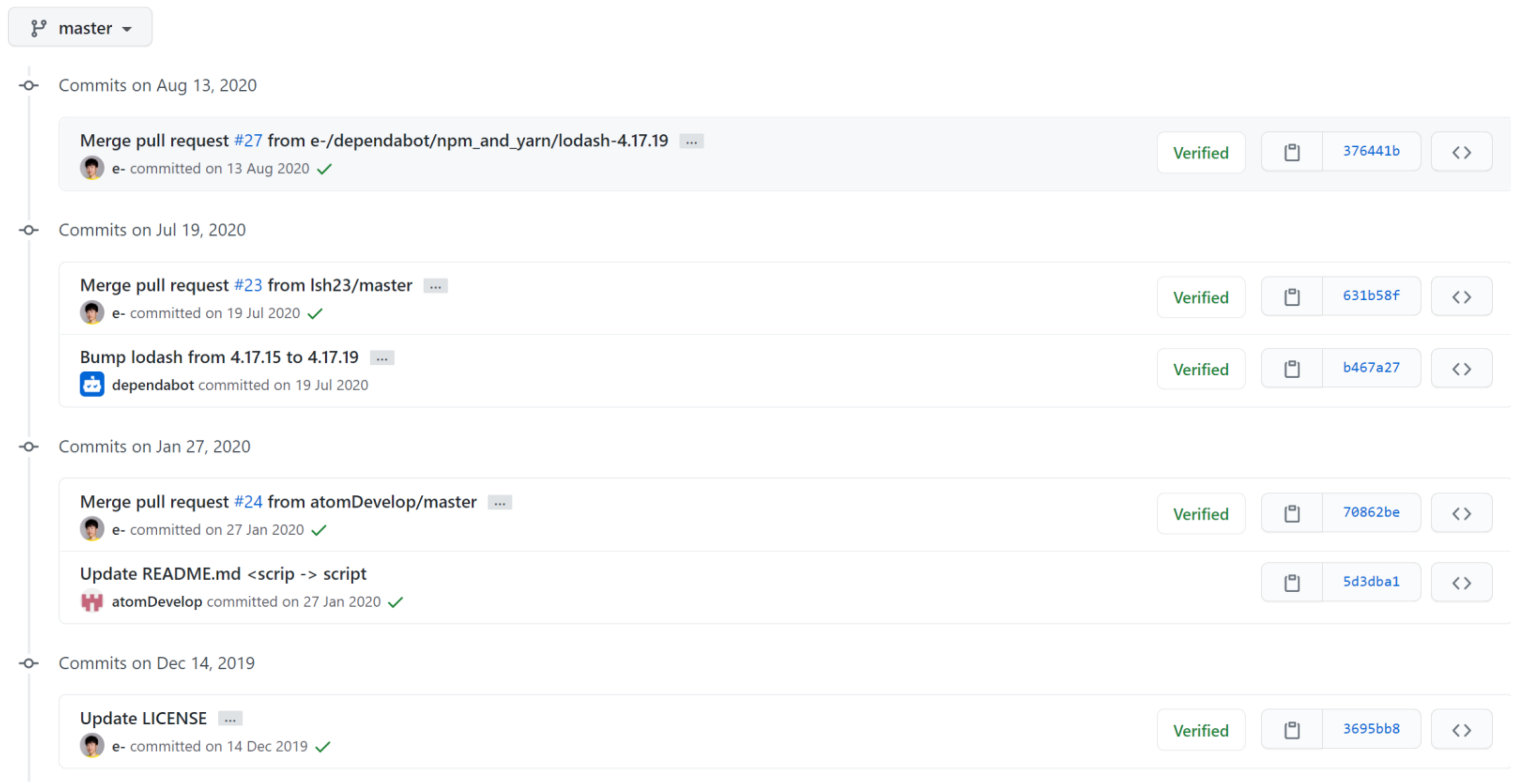
```
$ git push
Enter passphrase for key '/c/Users/jmjo/.ssh/id_rsa':
To github.com:e-/test.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'github.com:e-/test.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote 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)
```

Pushing to the Remote

- Once your commits are pushed to GitHub, you can list them on the GitHub website.



master

Commits on Aug 13, 2020

Merge pull request #27 from e-/dependabot/npm_and_yarn/lodash-4.17.19 …
e- committed on 13 Aug 2020 ✓ — Verified — 376441b

Commits on Jul 19, 2020

Merge pull request #23 from lsh23/master …
e- committed on 19 Jul 2020 ✓ — Verified — 631b58f

Bump lodash from 4.17.15 to 4.17.19 …
dependabot committed on 19 Jul 2020 — Verified — b467a27

Commits on Jan 27, 2020

Merge pull request #24 from atomDevelop/master …
e- committed on 27 Jan 2020 ✓ — Verified — 70862be

Update README.md <scrip -> script
atomDevelop committed on 27 Jan 2020 ✓ — 5d3dba1

Commits on Dec 14, 2019

Update LICENSE …
e- committed on 14 Dec 2019 ✓ — Verified — 3695bb8

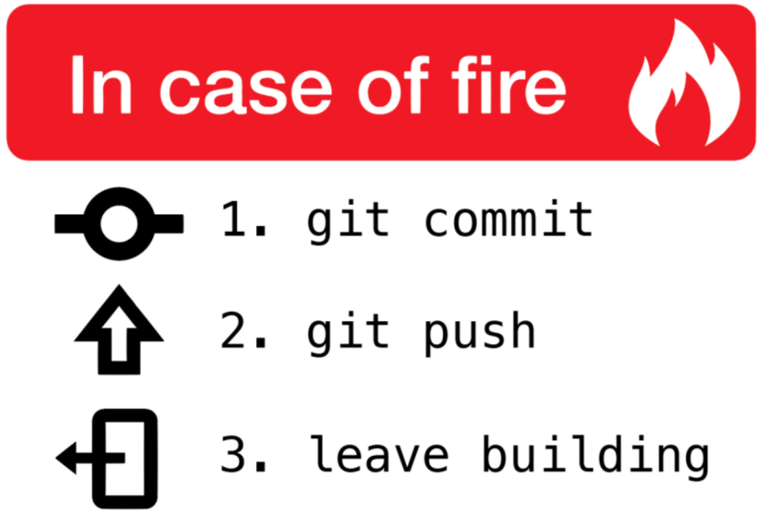
Pushing Every Commit?

- Not necessary, but recommended if
    1. you make changes on many files (so that others can *pull* the changes)
    2. you are working on the same file with your peers
    3. you want to make *others* to merge

- Personally, I usually push once every 6-7 commits.

Summary: 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
```