



# Distribute Source Code Management with Git Workshop





# Intro Git

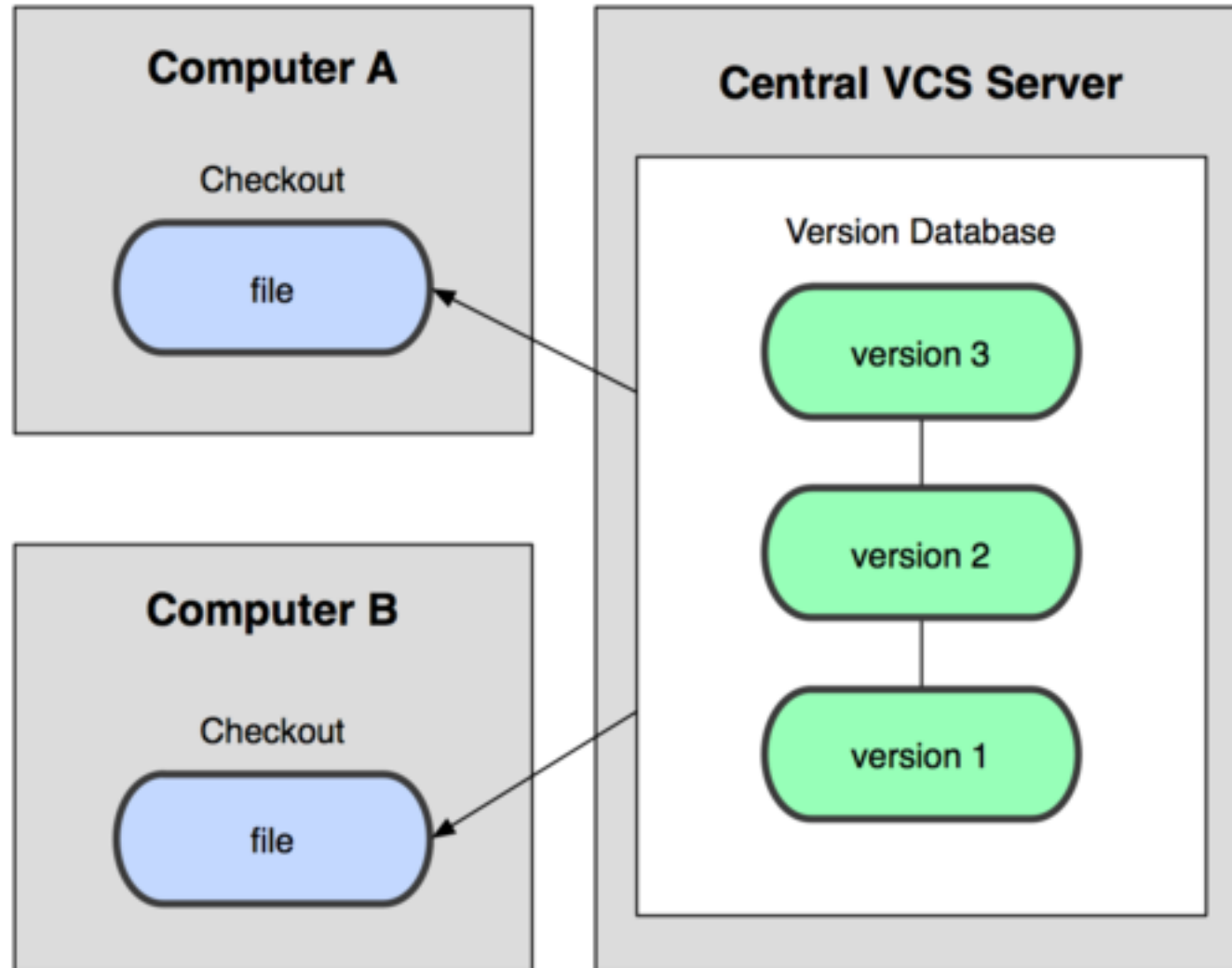
---

# Version Control

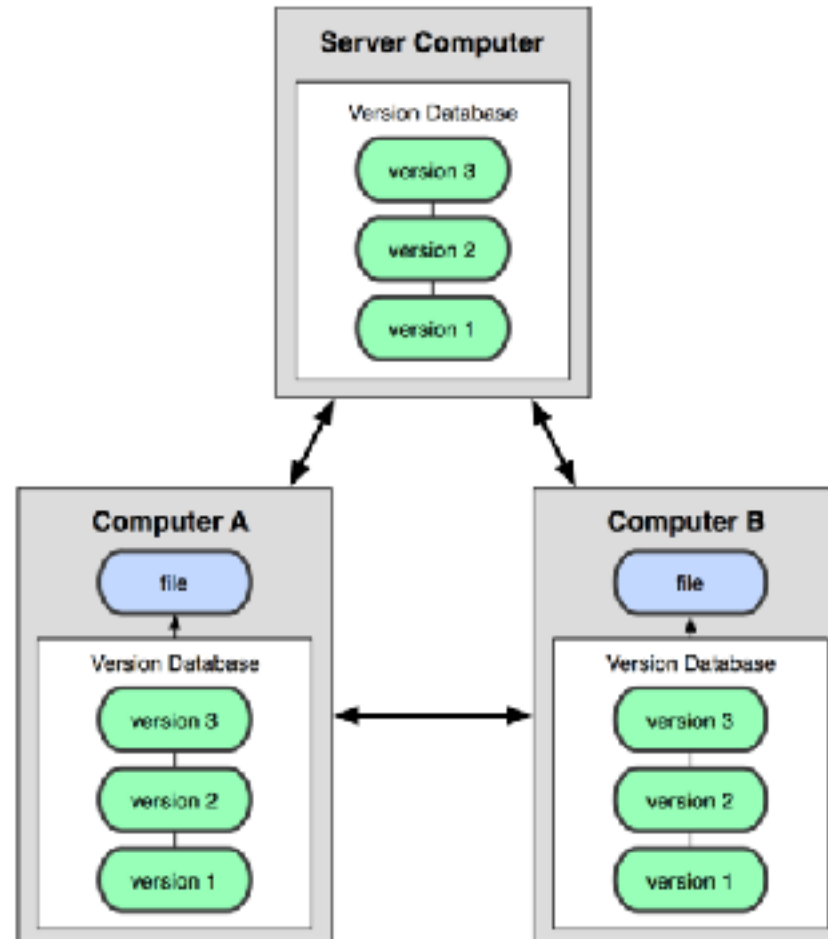
	Centralized	Distributed
Deltas	<div>subversion</div> <div>perforce</div>	<div>mercurial</div>
DAGs	<div>bitkeeper</div>	<div>git</div> <div>bazaar</div>



# Centralized Version Control



# Distributed Version Control



# เป้าหมายของการออกแบบ Git

**Speed**

**Simple design**

**Support for many parallel branches**

**Fully distributed**

**To handle large project like Linux kernel**



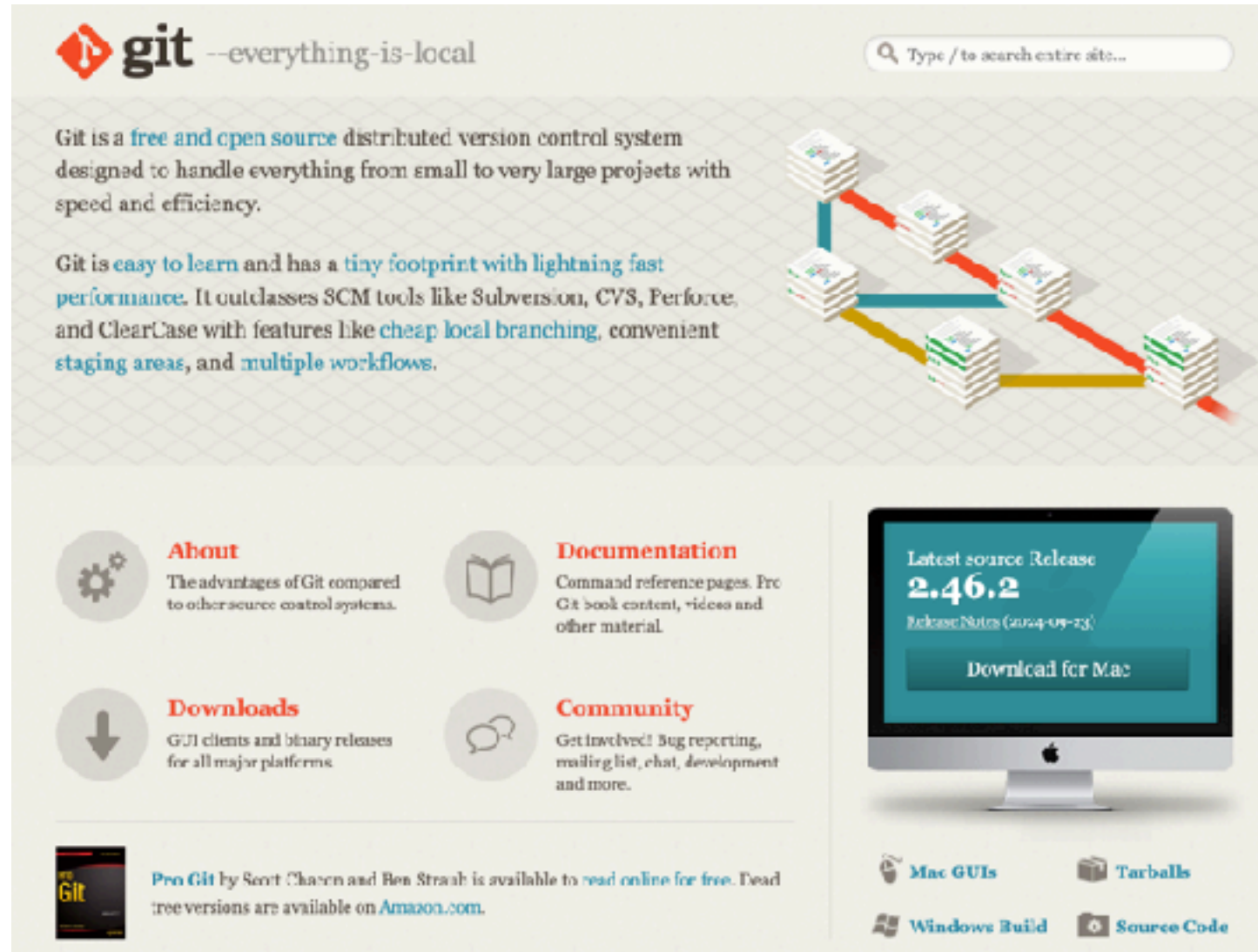


# Git Workflow

lorem ipsum dolor sit amet consectetur adipiscing elit  
maecenas porttitor congue massa fusce posuere magna  
sed pulvinar ultricies purus

[learn more](#)

# Install Git: <http://git-scm.com/>



The image shows the Git website homepage. At the top left is the Git logo (a red diamond with a white 'G') and the tagline "everything-is-local". To the right is a search bar with the placeholder text "Type / to search entire site...". Below the logo, there are two paragraphs of text. The first paragraph describes Git as a free and open source distributed version control system. The second paragraph describes Git as easy to learn and having a tiny footprint with lightning fast performance. To the right of the text is a diagram showing a network of stacks of books connected by lines, representing a distributed version control system. Below the text and diagram are four sections: "About" (with a gear icon), "Documentation" (with a book icon), "Downloads" (with a download arrow icon), and "Community" (with a speech bubble icon). Each section has a brief description. To the right of these sections is a large monitor displaying the latest source release "2.46.2" and a button to "Download for Mac". At the bottom left, there is a book cover for "Pro Git" and a link to read it online for free. At the bottom right, there are four links: "Mac GUIs", "Tarballs", "Windows Build", and "Source Code".

**git** --everything-is-local

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

**About**  
The advantages of Git compared to other source control systems.

**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

**Latest source Release**  
**2.46.2**  
Release Notes (2024-09-23)  
[Download for Mac](#)

**Pro Git** by Scott Chacon and Ben Straub is available to [read online for free](#). Head tree versions are available on [Amazon.com](#).

[Mac GUIs](#) [Tarballs](#)  
[Windows Build](#) [Source Code](#)





# Configuration Git on Local Machine

```
$git config --global user.name "your name"
```

```
$git config --global user.email "your email"
```

```
$git config -l
```



# Create a repository

Windows

Select git bash

Mac or  
Linux

Open Terminal

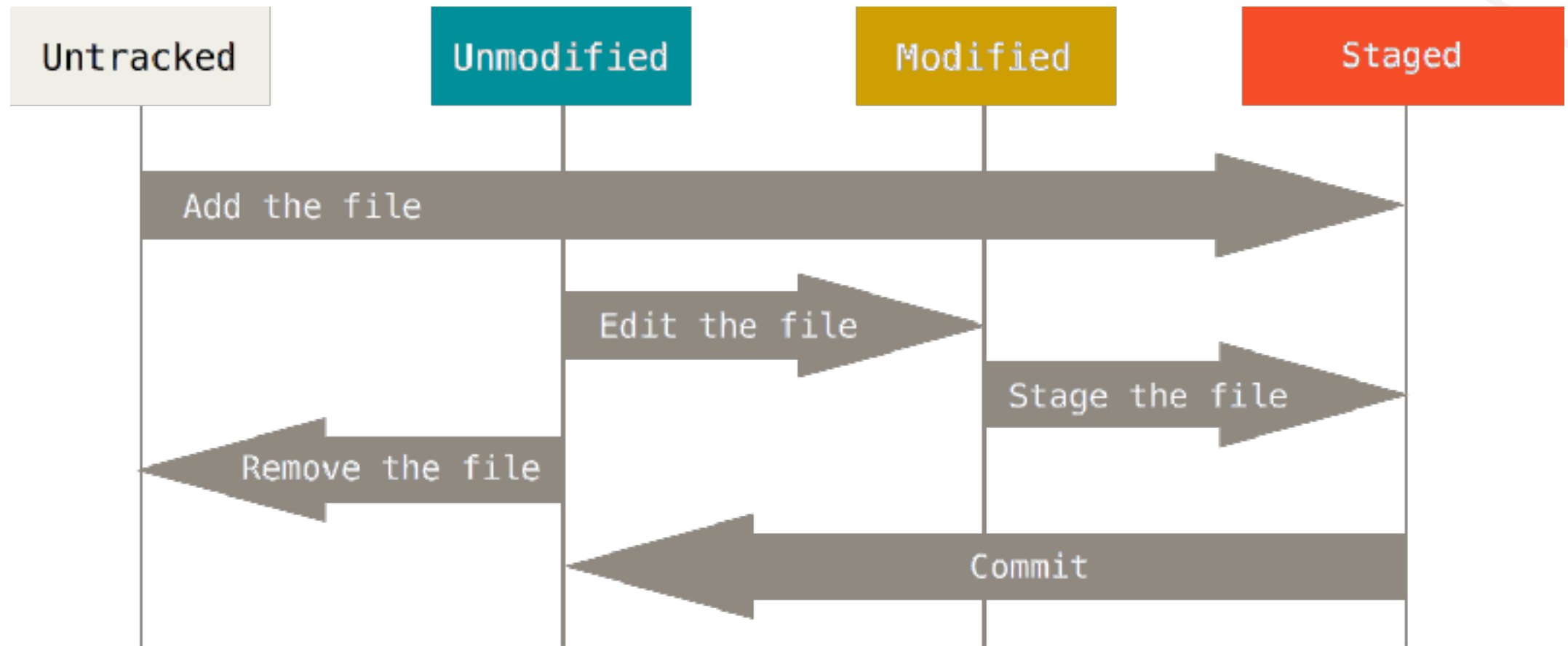
```
$mkdir git-workspace
```

```
$cd git-workspace
```

```
$git init
```



# Workflow



# Create a file and check the status

```
$touch README
```

```
$git status
```

```
On branch main
```

```
No commits yet
```

```
Untracked files:
```

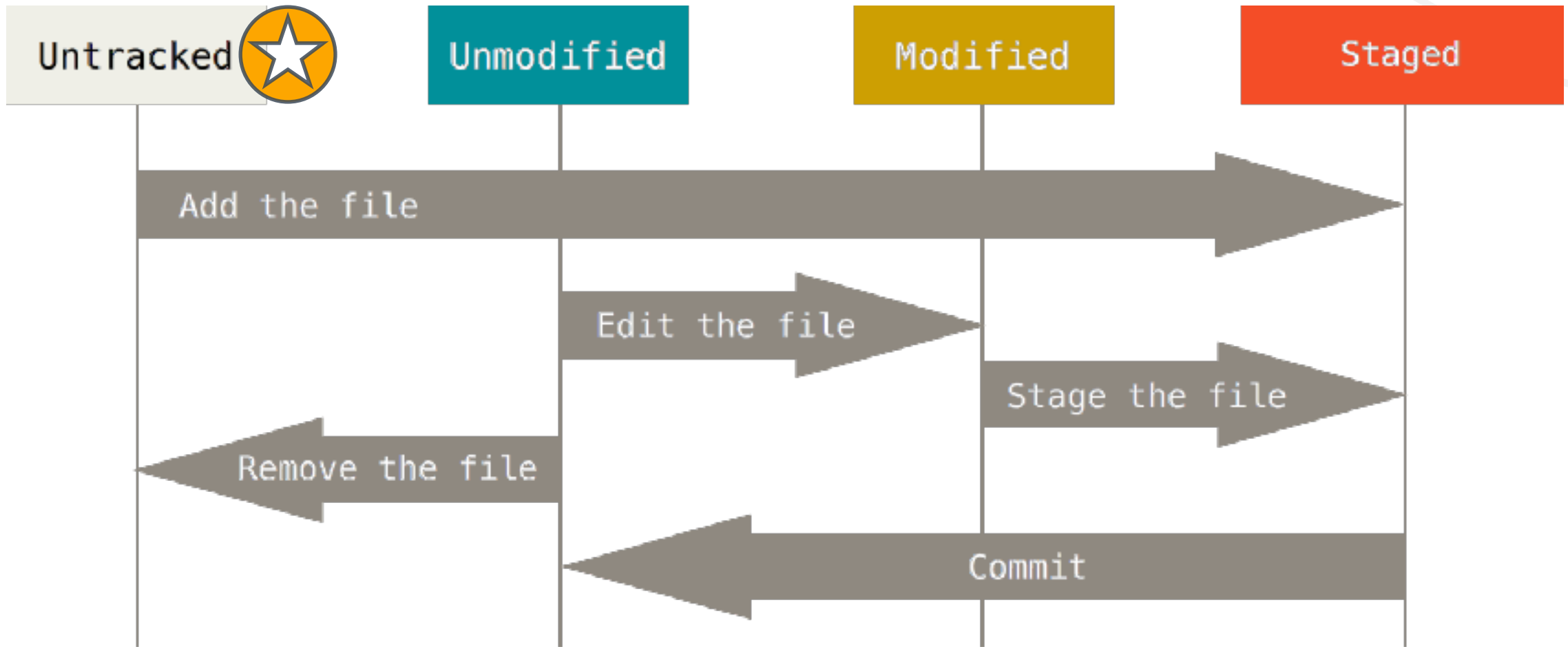
```
(use "git add <file>..." to include in what will be committed)
```

```
README
```

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



# Untracked



# Add the file and check the status

```
$git add README
```

```
$git status
```

On branch main

No commits yet

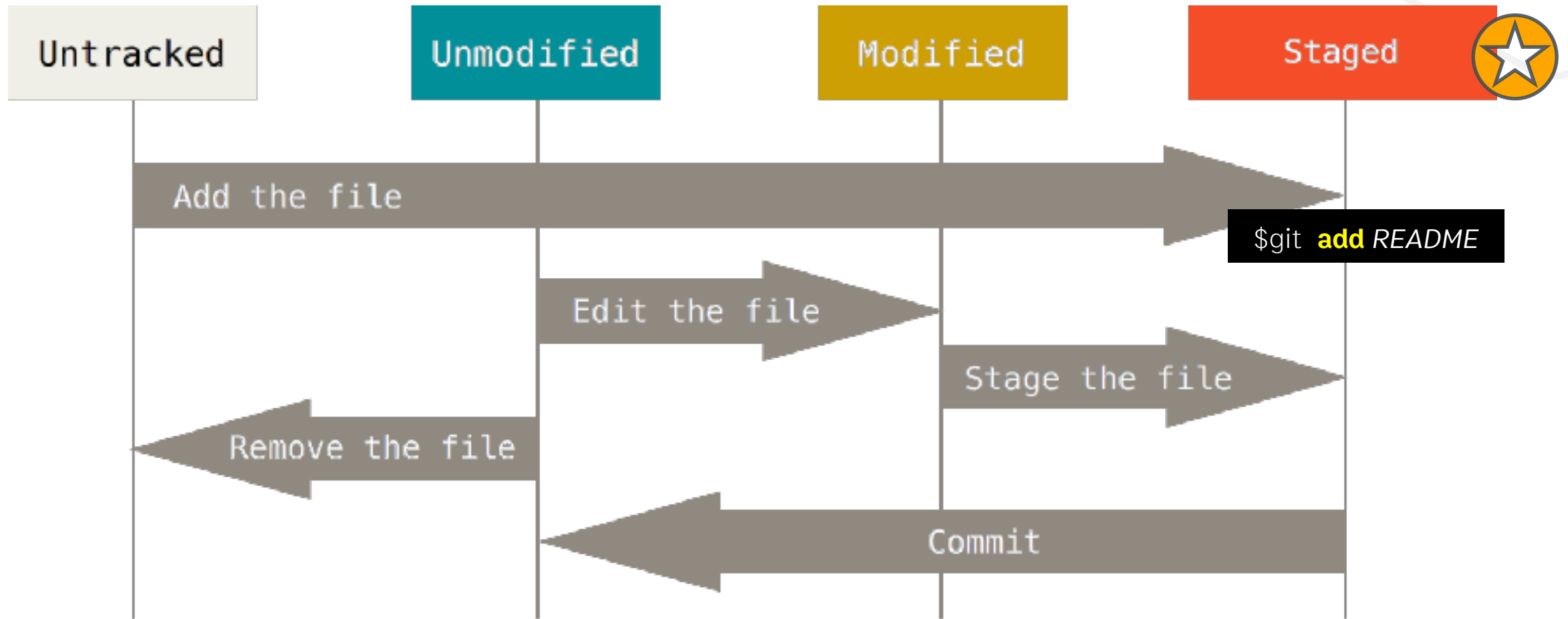
Changes to be committed:

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

new file: README



# Staged



# Commit your changes

```
$git commit -m "Message for this change"
```

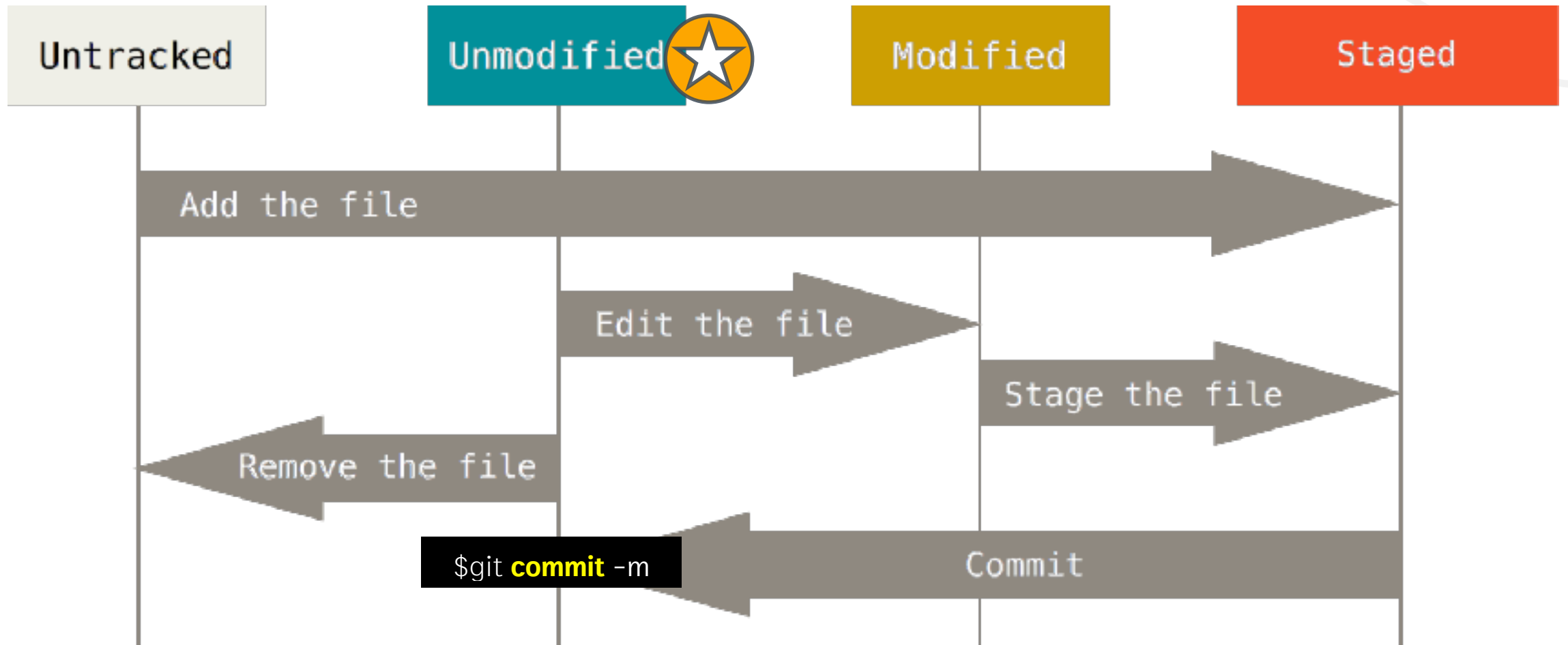
```
$git commit -m "Add README"
```

```
[main (root-commit) 779b411] Add README  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 README
```





# Unmodified



# Check the status

```
$git status
```

```
On branch main  
nothing to commit, working tree clean
```



# View staged change

```
$echo "test" > README
```

```
$git status
```

On branch main

Changes not staged for commit:

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

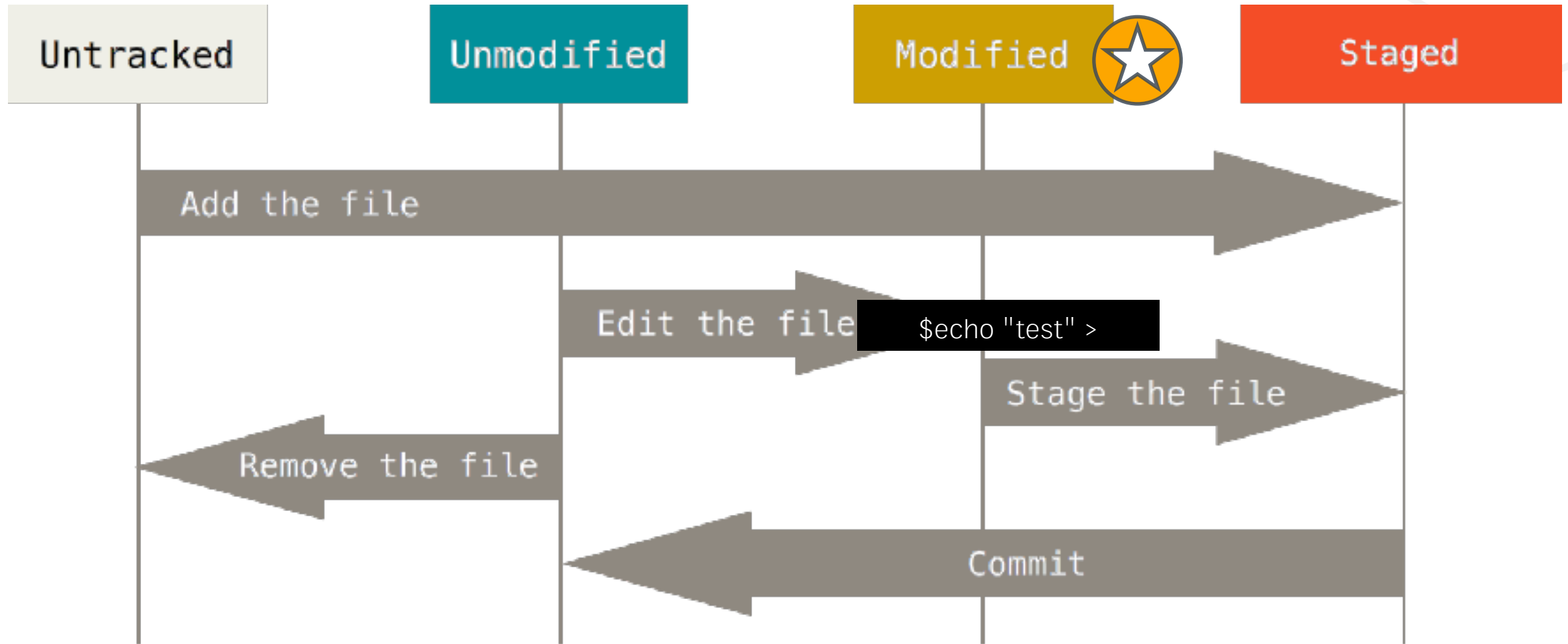
(use "git restore <file>..." to discard changes in working directory)

**modified: README**

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



# Modified



# Add the file and check the status

```
$git add README
```

```
$git status
```

On branch main

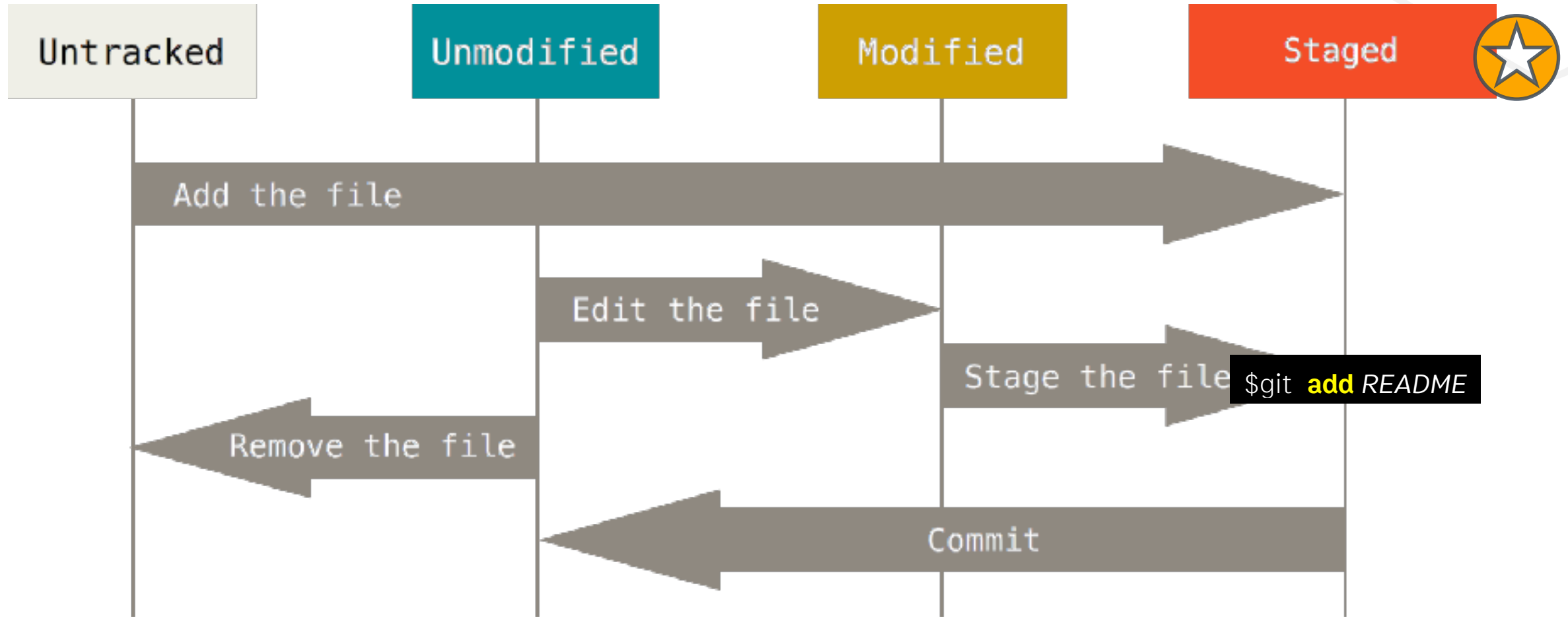
Changes to be committed:

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

modified: README



# Staged



# Commit your changes

```
$git commit -m "Edit README"
```

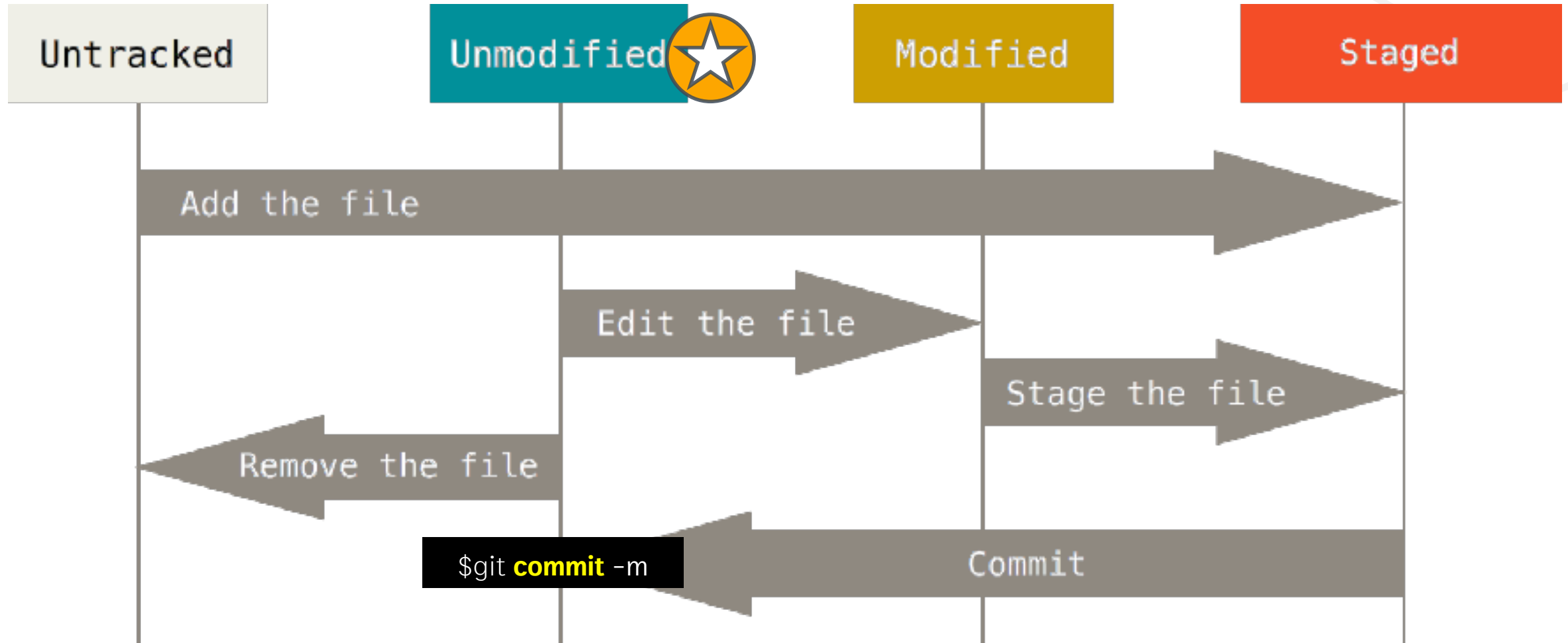
```
[main 656bbdd] Edit README  
1 file changed, 1 insertion(+)
```

```
$git status
```

```
On branch main  
nothing to commit, working tree clean
```



# Unmodified





# Ignore files and folders

```
$touch note
```

```
$mkdir tmp
```

```
$touch tmp/file
```

```
$git status
```

```
$touch .gitignore
```

```
$echo "note" > .gitignore
```

```
$echo "tmp/" >> .gitignore
```



# Modified README

```
$echo "# hello" > README
```

```
$git add README
```

```
$git commit -m "Edit README content"
```

```
$git status
```

```
On branch main  
nothing to commit, working tree clean
```



# Git reset

```
$git reset <mode> <commit ID>
```

```
$git reset --soft <commit ID>
```

```
$git reset --mixed <commit ID>
```

```
$git reset --hard <commit ID>
```



git log \_

```
$git log
```

```
$git log --oneline
```

```
$git log --oneline --graph
```



# Git reset

Unmodified  
493ec5b

test

Modified

# hello

Stage

# hello

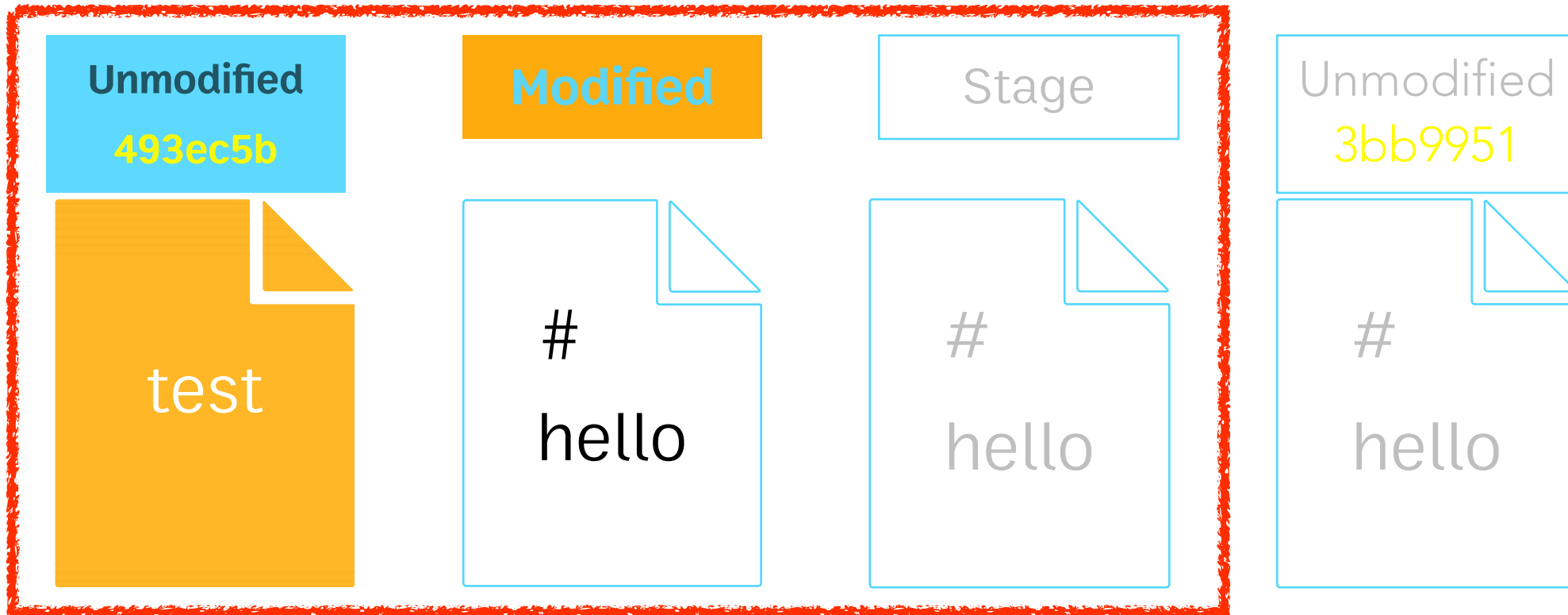
Unmodified  
14a5744

# hello



# git reset --soft

```
$git reset --soft 493ec5b
```



git reset --mixed [default]

```
$git reset 493ec5b
```

```
$git reset --mixed 493ec5b
```

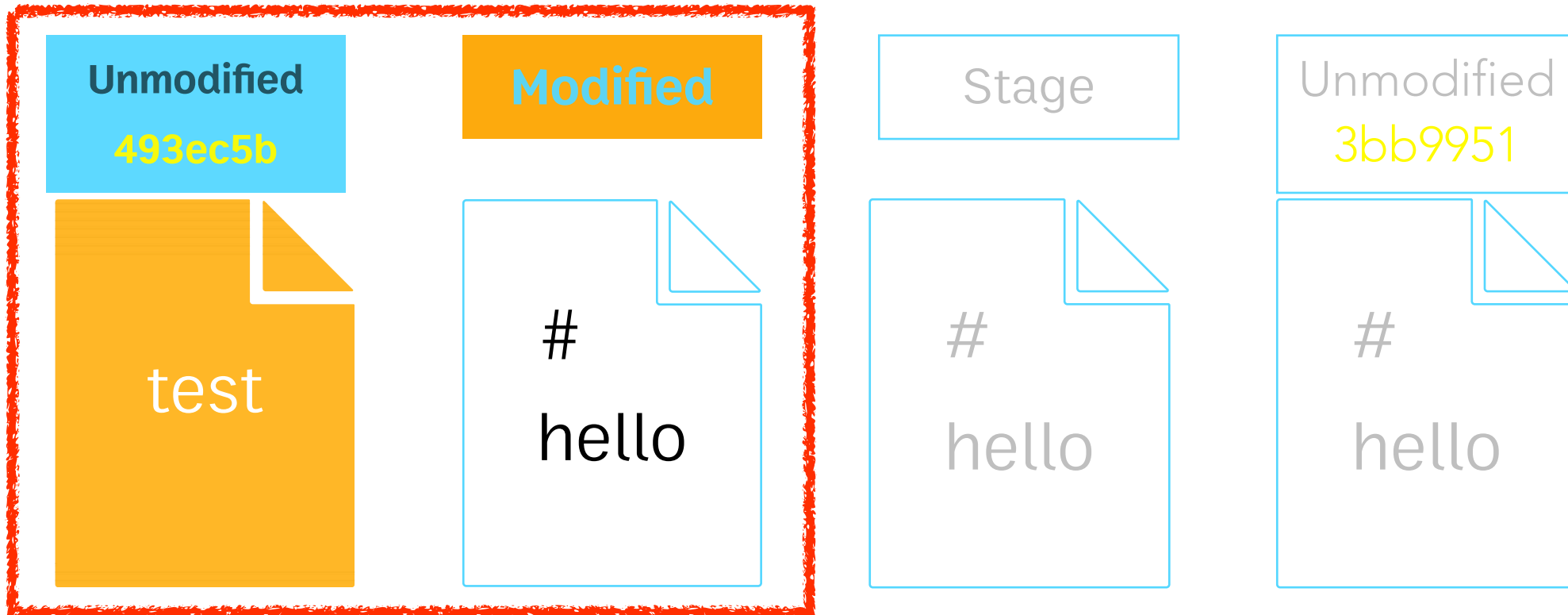
Unstaged changes after reset:

M README



# git reset --mixed [default]

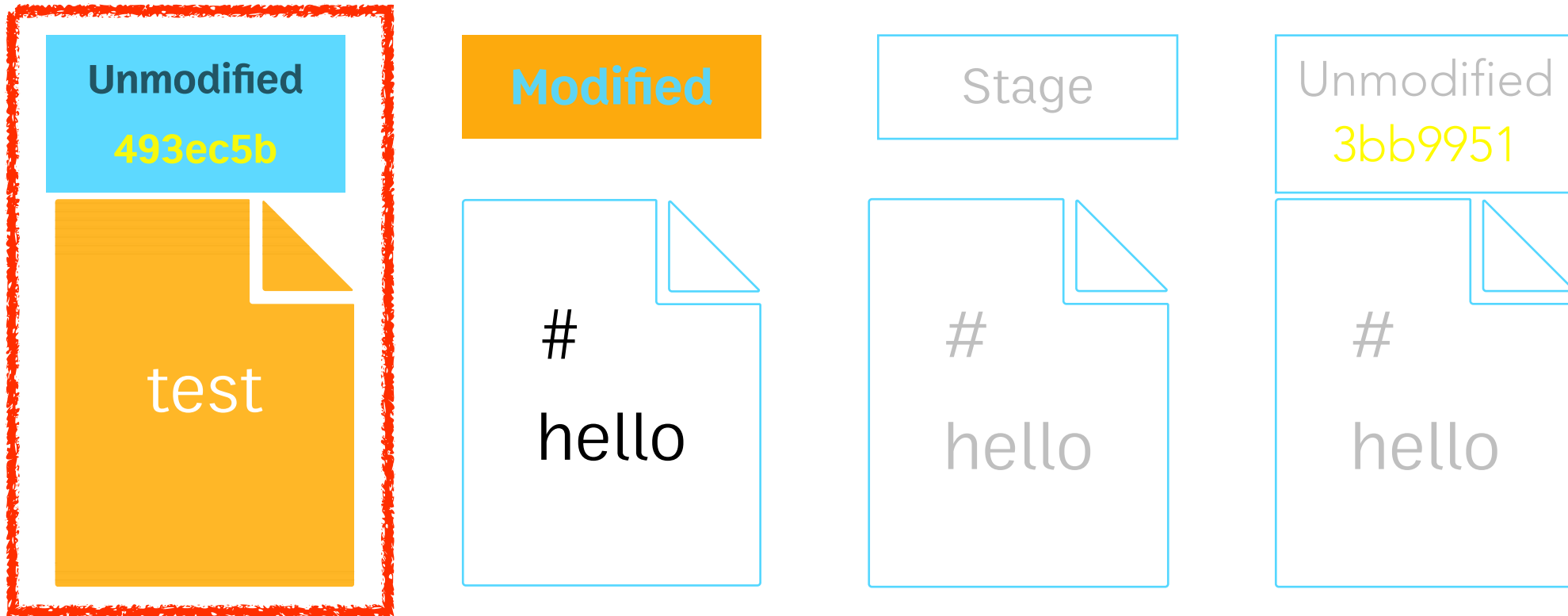
```
$git reset 493ec5b
```





# git reset --hard

```
$git reset --hard 493ec5b
```



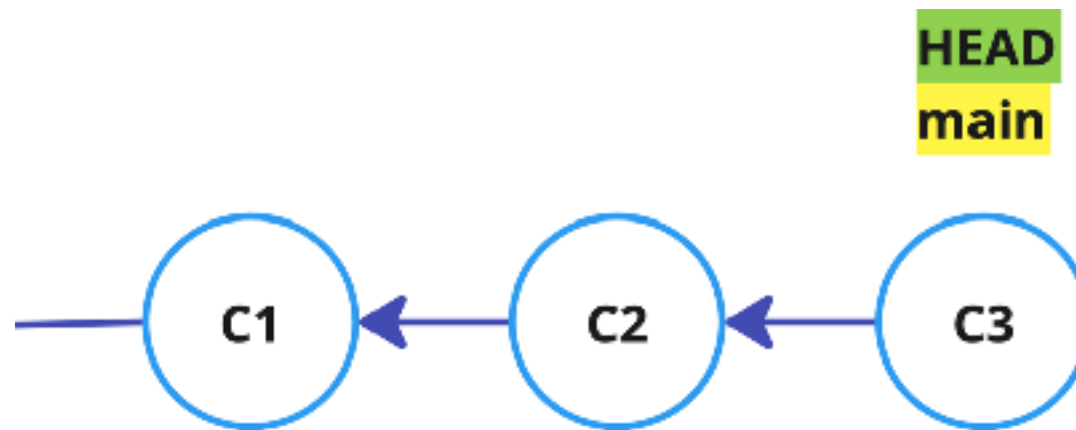


# Git Branch

---

# Current branch

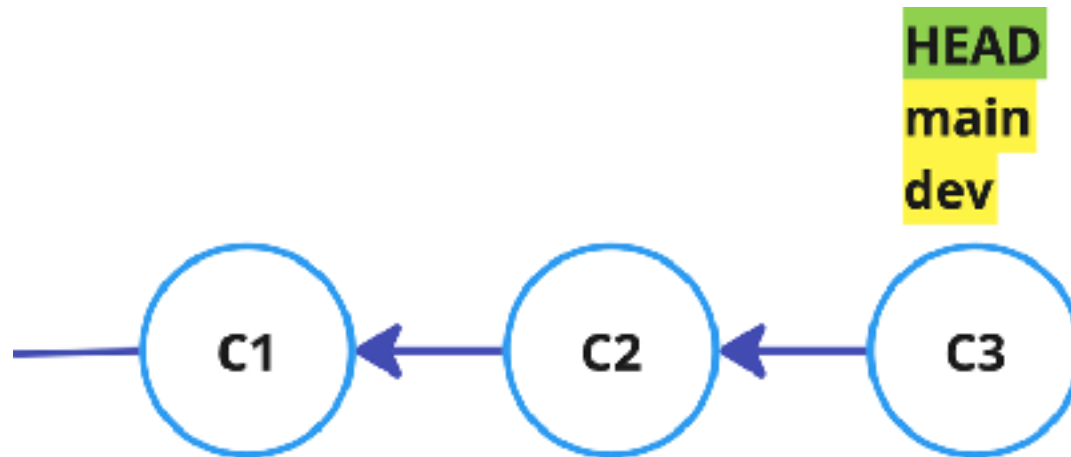
\$git **branch**



# Create new branch

```
$git branch <BRANCH NAME>
```

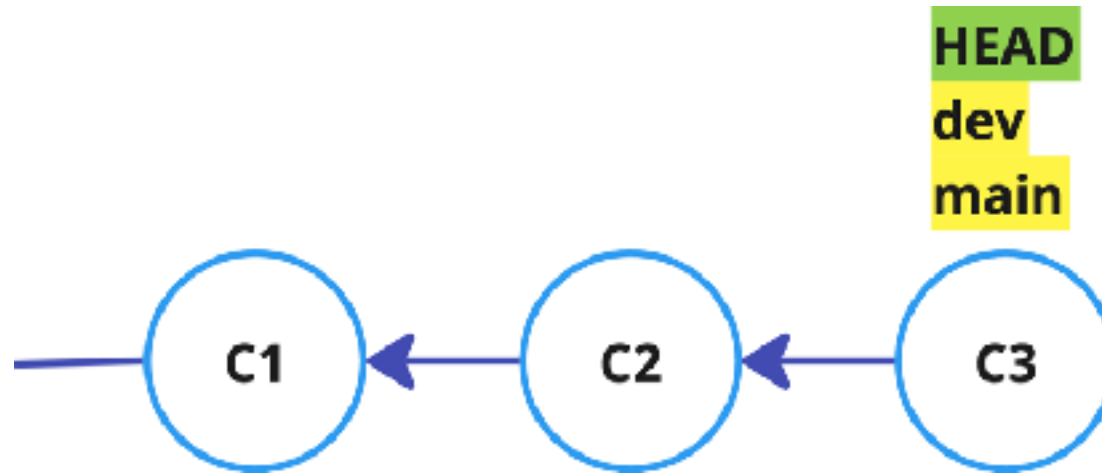
```
$git branch dev
```



# Switch branch

```
$git switch <BRANCH NAME>
```

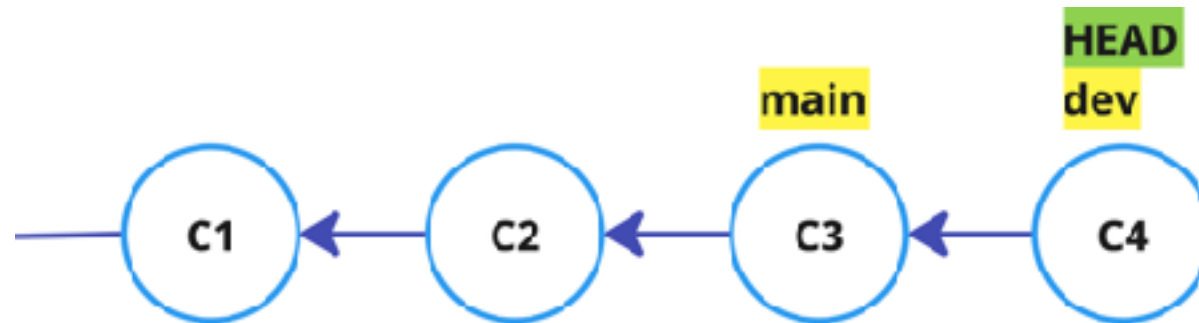
```
$git switch dev
```



# Modified and commit on dev

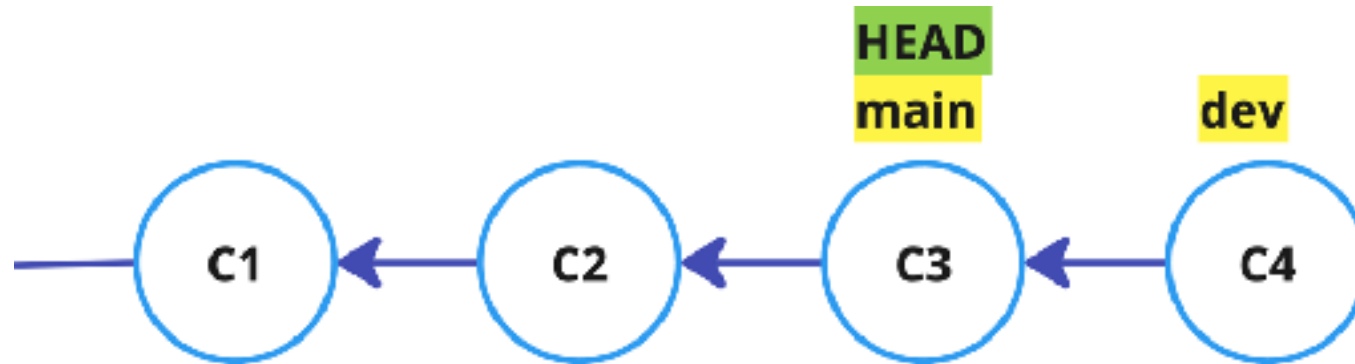
```
$echo "On branch dev" > README
```

```
$git commit -am "Add new content on branch dev"
```



# Switch to master

```
$git switch main
```



# Create and Switch branch

```
$git switch -c BRANCH NAME
```

```
$git branch BRANCH NAME
```

```
$git switch BRANCH NAME
```

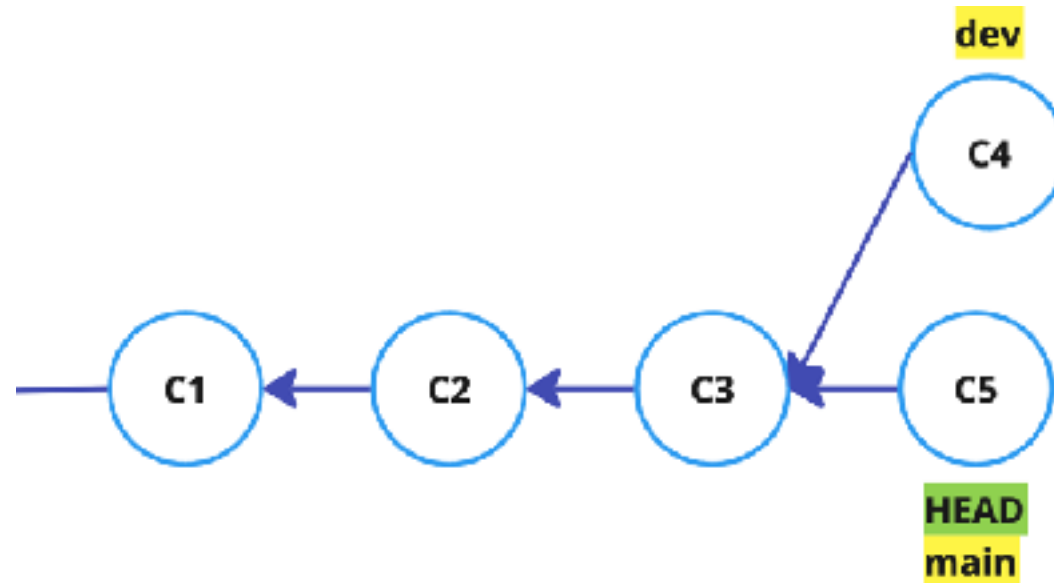




# Modified and commit on main

```
$echo "On branch main" >> README
```

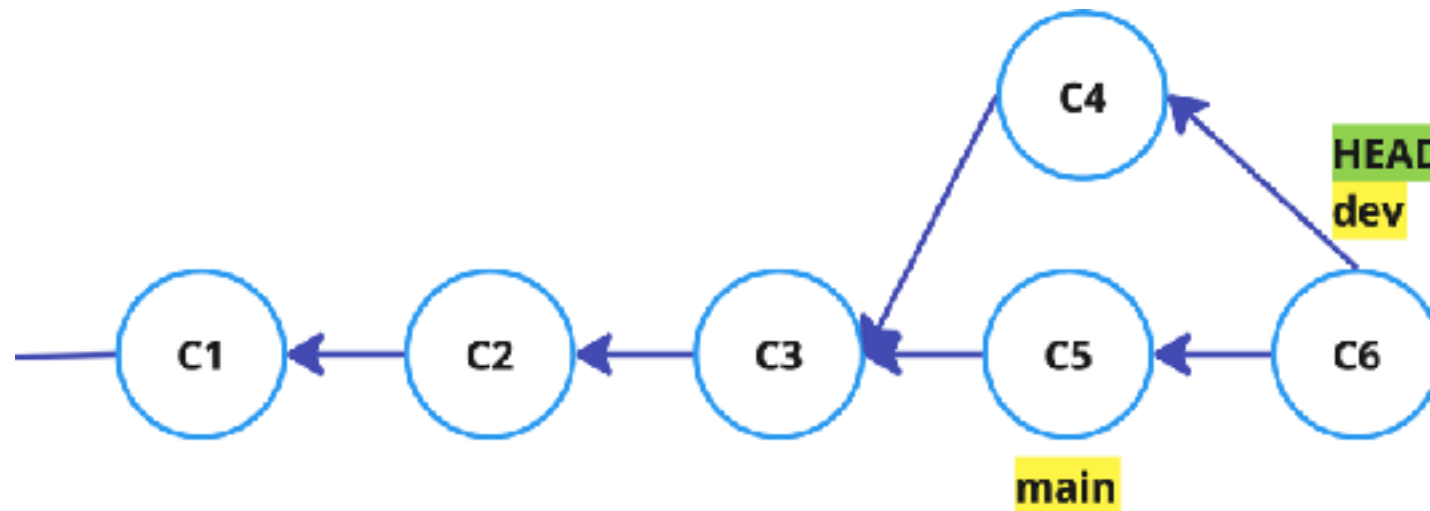
```
$git commit -am "Add new content on branch main"
```



# Merge

```
$git switch dev
```

```
$git merge main
```



# Conflict

```
$git merge main
```

```
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
```

```
$git status
```

```
On branch dev
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:   README
```

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



# Conflict

## README

```
<<<<<<< HEAD
On branch dev
=====
# hello
On branch main
>>>>>>> main
```



# Fix conflicts

```
git status
```

```
git add README
```

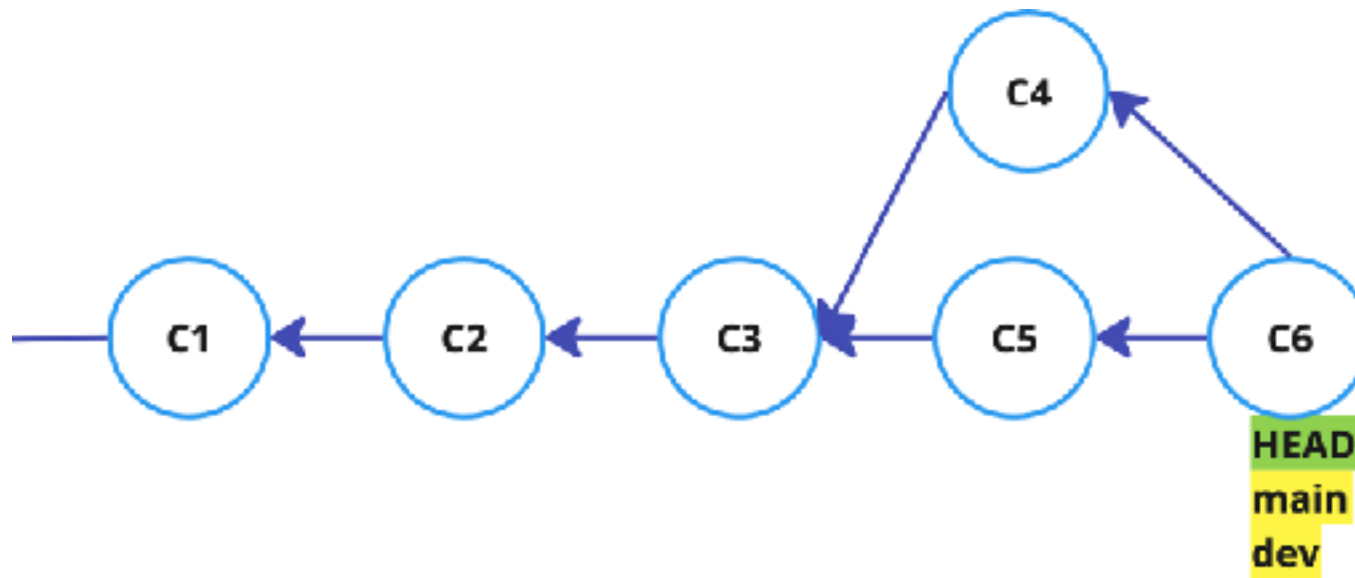
```
git commit
```



# Merge

```
$git switch main
```

```
$git merge dev
```



# Avoid Merge Conflict

**Small change and commit**

**Early merge**

**Single Responsibility Principle**

**Communication is a Key**

**Mob programming**



# Rebase



Allows you to write history differently  
Cleaner history  
rebase -i (interactive)





# Reset to before merge: dev, main

```
git log --oneline --graph --all
```

```
copy dev commit ID(before merge)
```

```
copy main commit ID(before merge)
```



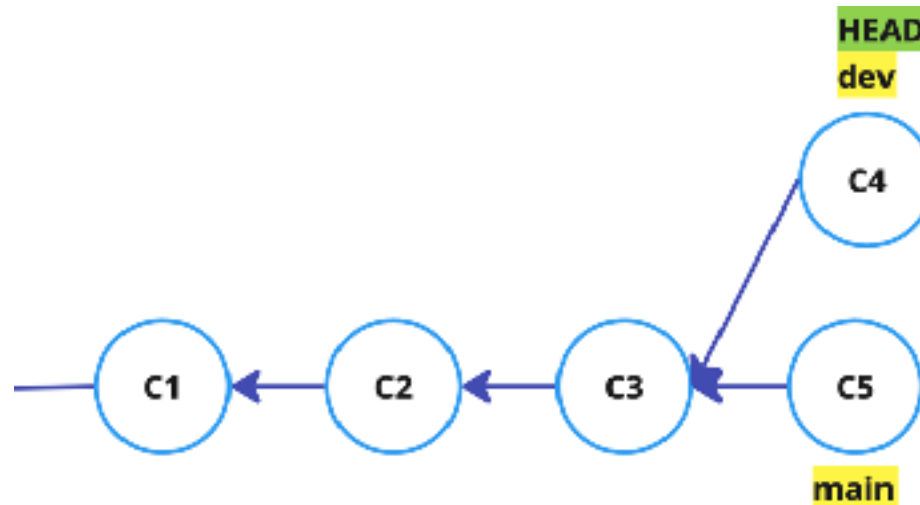
# Reset to before merge: dev, main

```
git switch main
```

```
git reset --hard commit ID
```

```
git switch dev
```

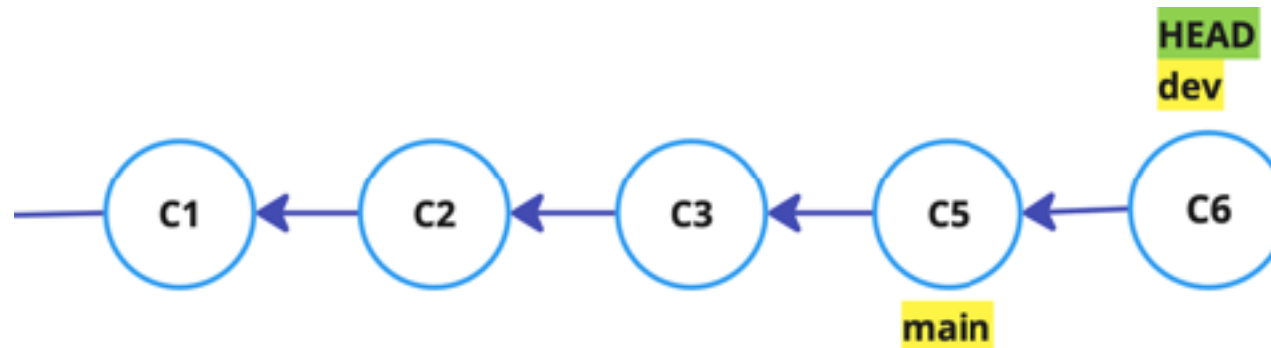
```
git reset --hard commit ID
```



# Rebase

```
$git checkout dev
```

```
$git rebase main
```



# data infographics

Auto-merging README

CONFLICT (content): Merge conflict in README

error: could not apply dffbb9f... Add new content on branch dev

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 rebase --abort".

hint: Disable this message with "git config advice.mergeConflict false"

Could not apply dffbb9f... Add new content on branch dev

##### HEAD

# hello

On branch main

=====

On branch dev

>>>>>> dffbb9f (Add new content on branch dev)

README: needs merge

You must edit all merge conflicts and then  
mark them as resolved using git add



# Tag

---

## Show tags

```
git tag
```

## Add new tag

```
git tag <tag_name>
```

```
git tag -a <tag_name> <commit_id>
```

```
git tag -am "Reason for tag" <tag_name> <commit_id>
```

## Remove tag

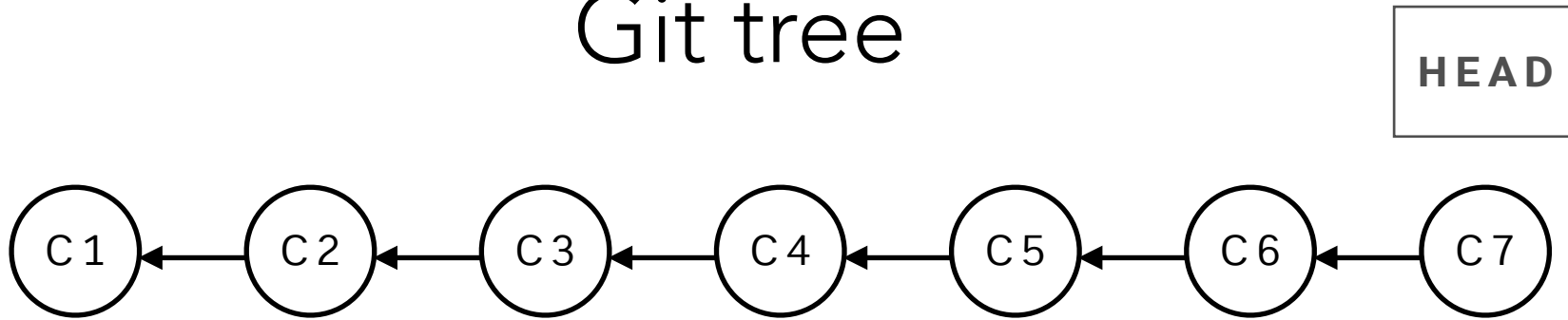
```
git tag -d <tag_to_remove>
```

## Remove all tag

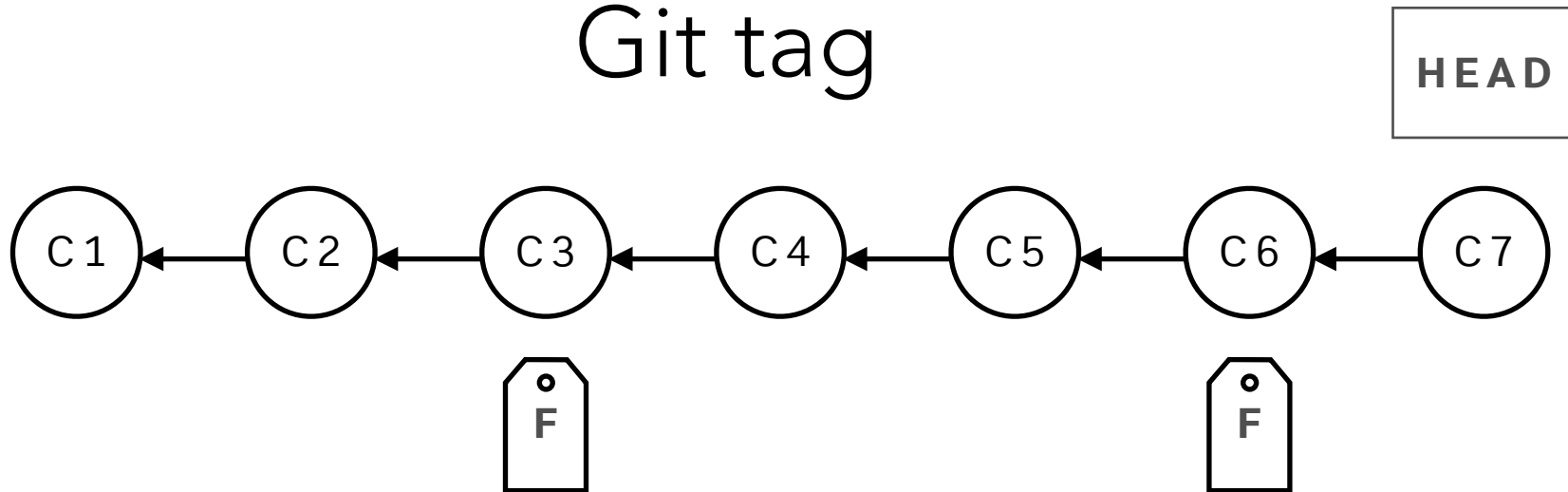
```
git tag | xargs -n1 git tag -d
```



## Git tree



## Git tag





ETC,...



# Commit message convention

- **build**: Build related changes (eg: npm related/ adding external dependencies)
- **chore**: A code change that external user won't see (eg: change to .gitignore file or .prettierrc file)
- **feat**: A new feature
- **fix**: A bug fix
- **docs**: Documentation related changes
- **refactor**: A code that neither fix bug nor adds a feature.  
(eg: You can use this when there is semantic changes like renaming a variable/ function name)
- **perf**: A code that improves performance
- **style**: A code that is related to styling
- **test**: Adding new test or making changes to existing test

<https://dev.to/ishanmakadia/git-commit-message-convention-that-you-can-follow-1709>





# Git Stash

## Show Stash

```
git stash list
```

## Stash Uncommit

```
git stash
```

```
git stash -m <message>
```

## Unstash

```
git stash pop
```

```
git stash pop -n stash@{<index>}
```



# Be careful

## **Force Push**

`git push -f`

## **Reset Remote Branch**

`git reset [--hard, --mixed, --soft] then git push`

## **Pull Rebase(not frequency sync)**

`git pull --rebase`

`## Remove all tag`

`git tag | xargs -n1 git tag -d`





# Remote Repository

---

# Working with remote repository

**git clone**

VS

**git remote add**



# git remote

## Quick setup — if you've done this kind of thing before

 Set up in Desktop

or

**HTTPS** **SSH** `git@github.com:<account name>/hello-world.git` 

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).


## ...or create a new repository on the command line

```
echo "# hello-world" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:<account name>/hello-world.git
git push -u origin main
```



## ...or push an existing repository from the command line

```
git remote add origin git@github.com:<account name>/hello-world.git
git branch -M main
git push -u origin main
```

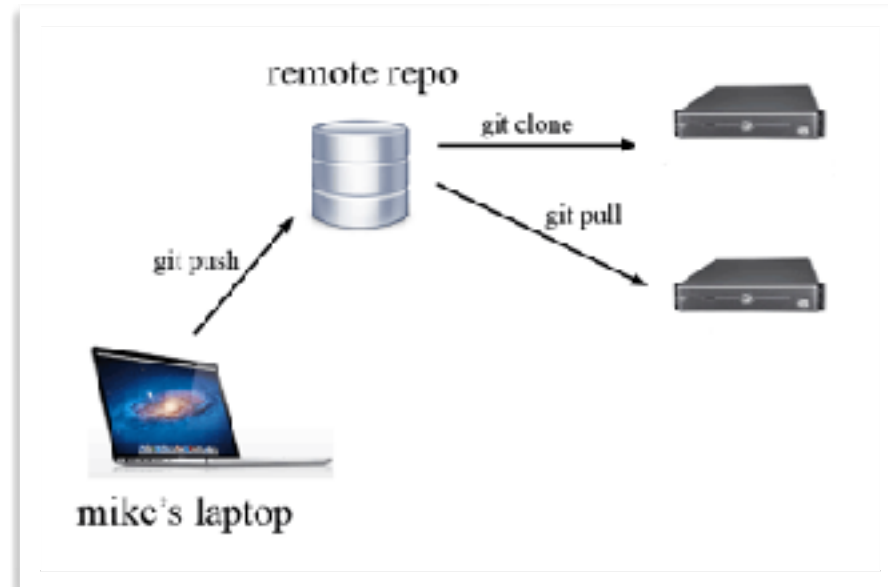


# git clone

```
$git clone git-url
```

```
$git clone git@github.com:<account name>/hello-world.git
```

```
$git clone https://github.com/<account name>/hello-world.git
```



# git push, git pull

## **Share code to remote**

```
git push -u origin <branch-name>
```

```
git push
```

```
git push origin <branch-name>
```

## **Share local branch to remote**

```
git push origin <branch-name>
```

## **Remove branch from remote**

```
git push origin <:branch-name>
```

```
git push origin -d <branch-name>
```

## **Pull code from remote**

```
git pull origin <branch-name>
```

```
git pull
```

## **Pull code from remote with rebase**

```
git pull --rebase
```



# Share tag

## **Share tags to remote**

```
git push <remote_name> --tags
```

```
git push <remote_name> <tag_name>
```

## **Remove tag on remote**

```
git push <remote_name> :<tag_to_remove>
```

```
git push --delete <remote_name> <tag_to_remove>
```

## **Fetch tags from remote**

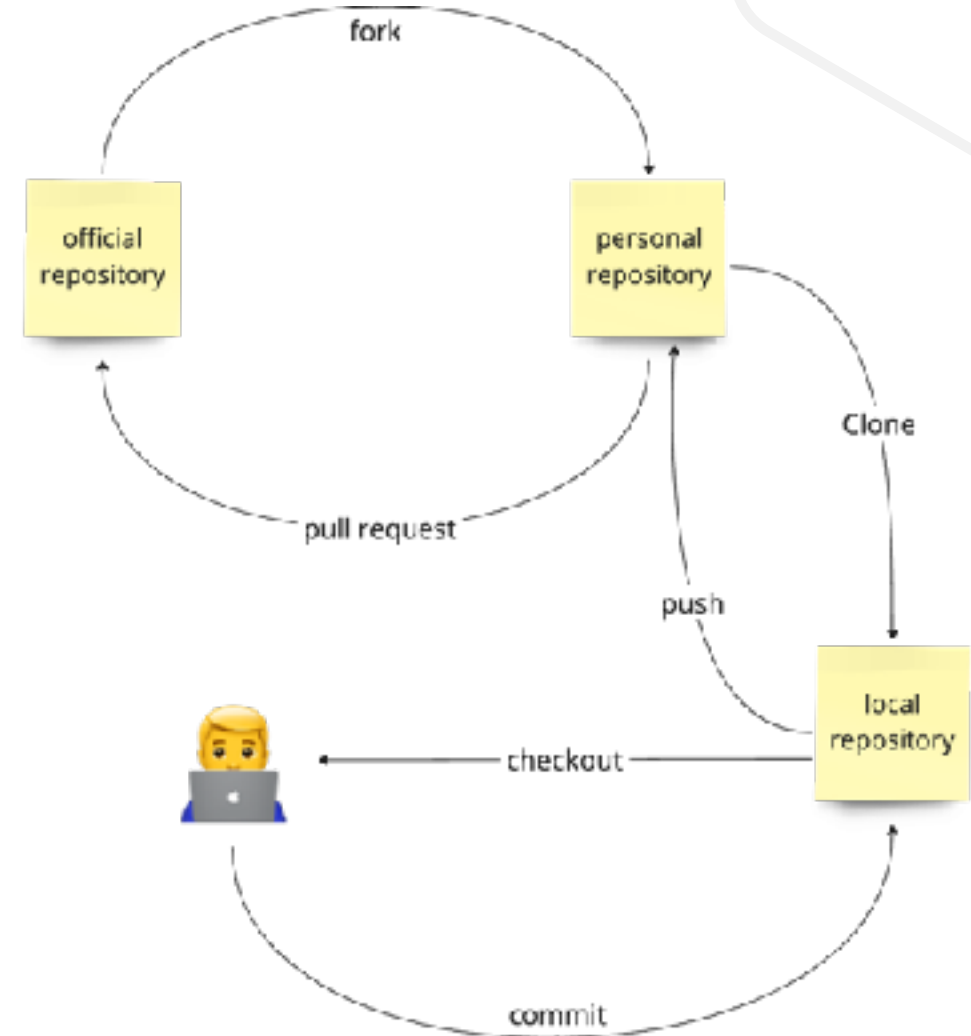
```
git fetch --all --tags
```





# Pull Request

A **pull request (PR)**, also known as a **merge request**, is a fundamental concept in software development, particularly when using version control systems like Git. It serves as a proposal to merge code changes from one branch into another, typically from a feature branch into the main branch of a project.

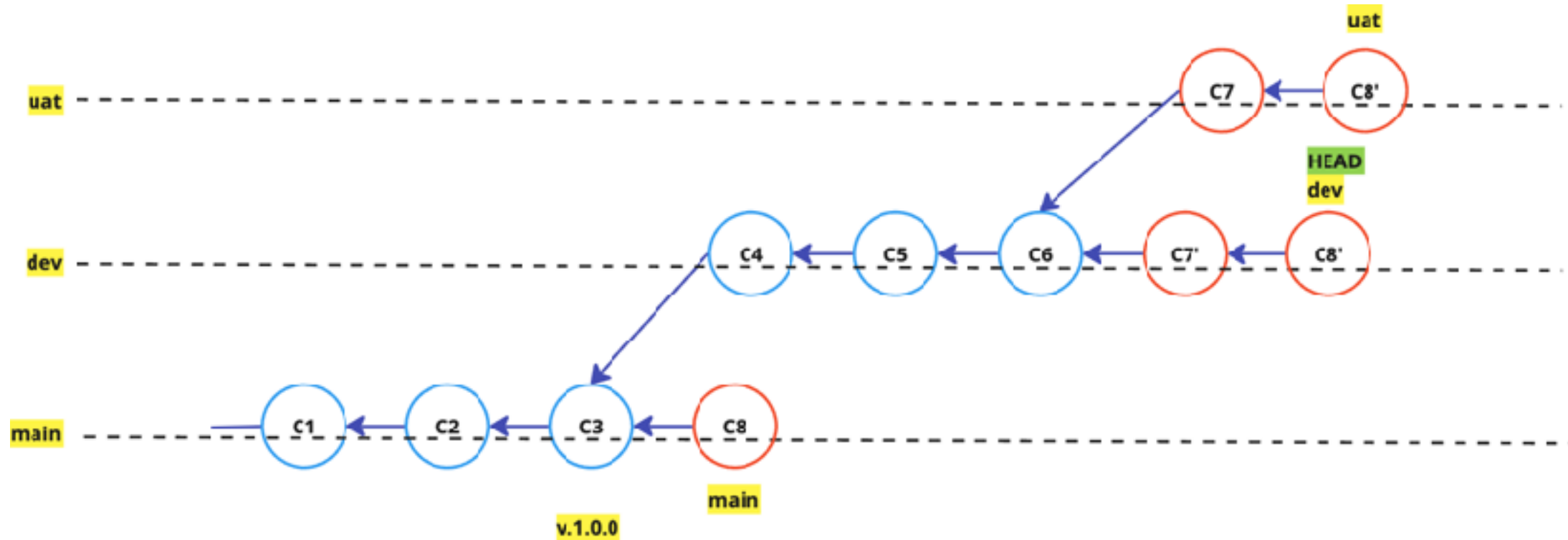




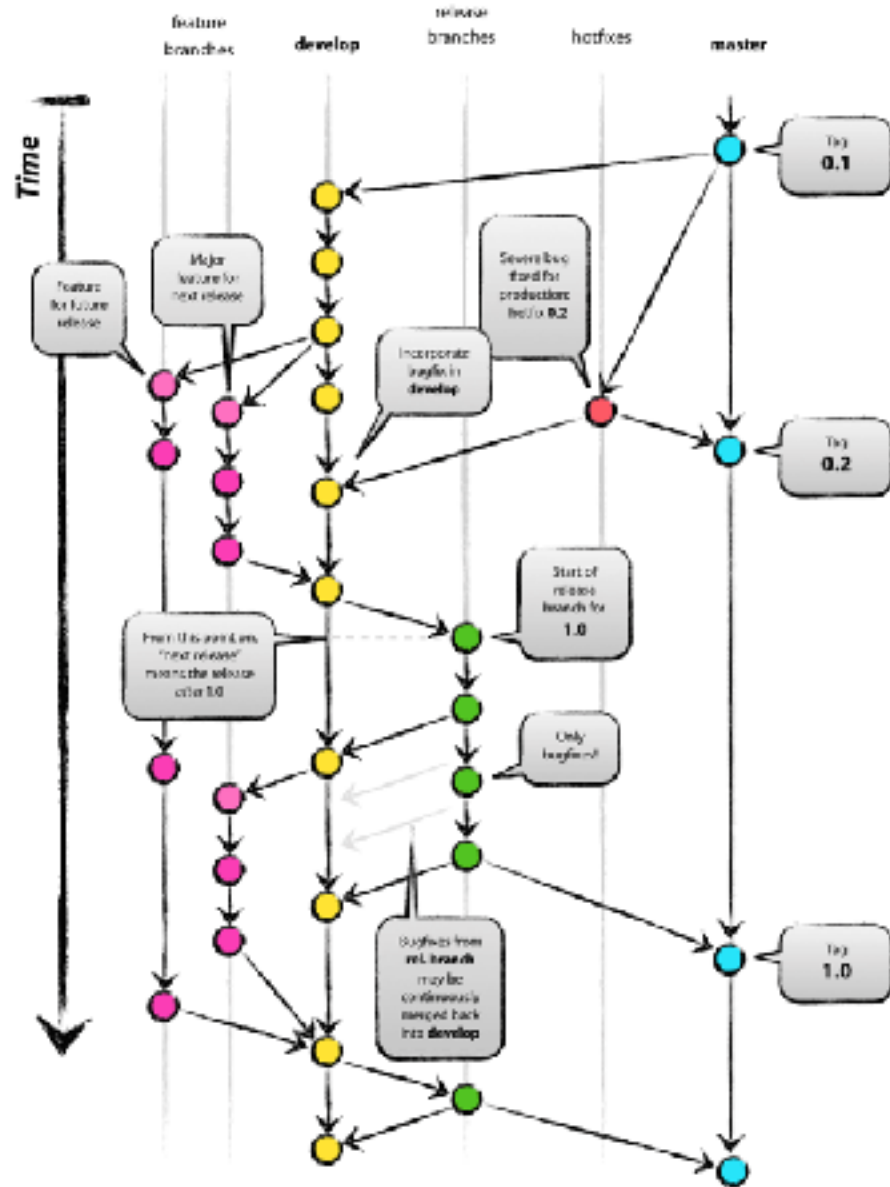
# Branch Strategy

---

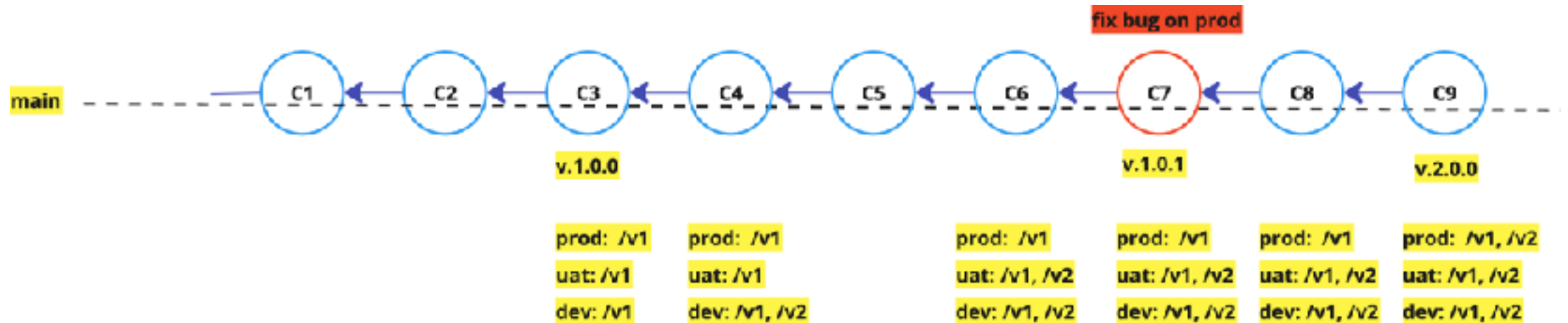
# Branch by Environment



# Git flow



# Feature Flag



# Branch for Release





# thanks