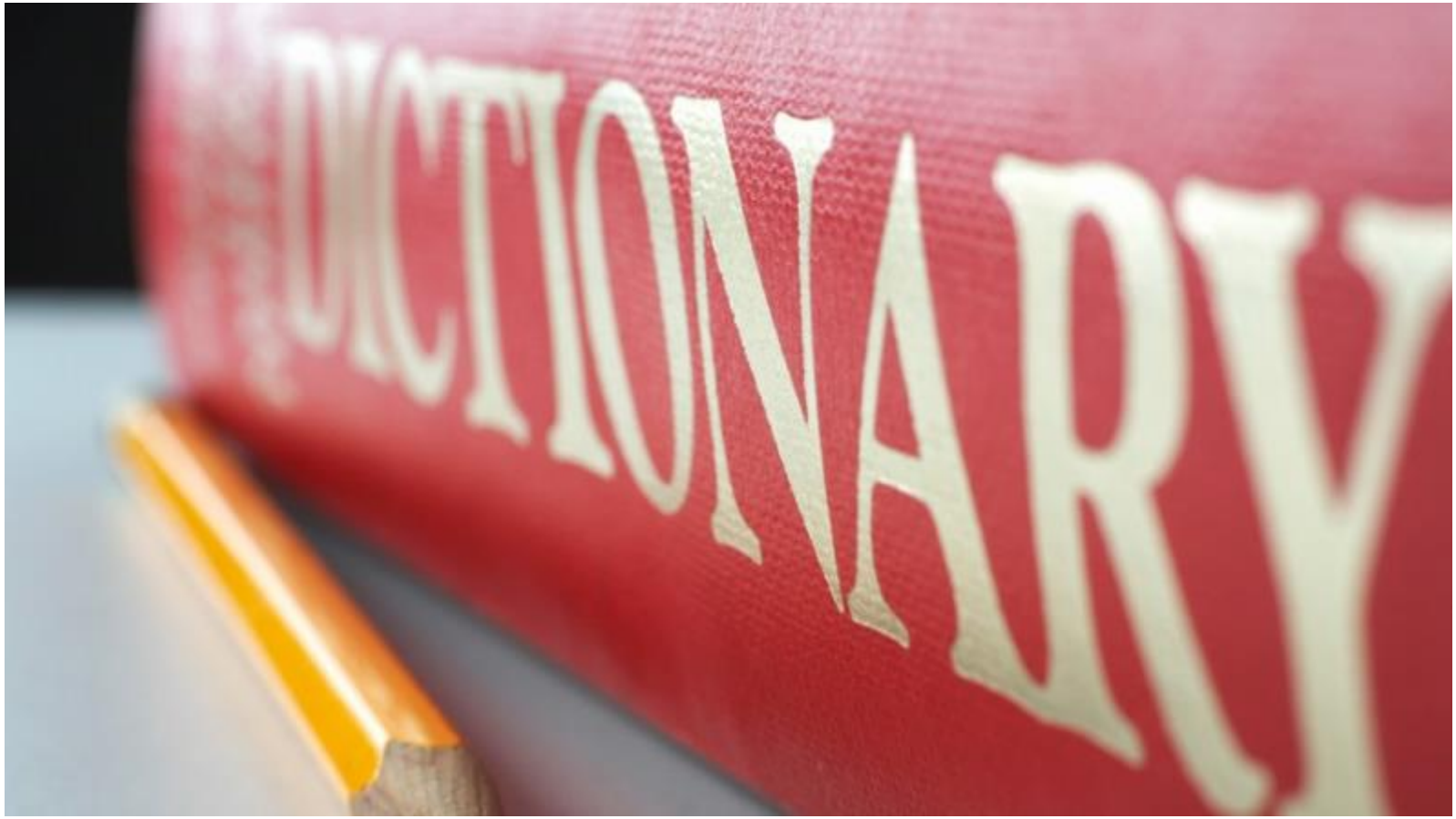
Git for Developers

Concepts
Hands-on exercises
Command-line
UI clients

Agenda

- What we will learn
  - Terminology
  - Basic operations
  - Changes, staging, commit
  - Merge and conflict resolution

Section - Terminology



What is Version Control?

- Management of changes
- Why is it important?
  - Revert code changes
  - Never loose code
  - Maintain multiple versions of a product
  - See the difference between two (or more) versions of your code
  - Prove that a particular change broke or fixed a piece of code
  - Review the history of some code
  - Submit a change to someone else's code
  - Share your code, or let other people work on your code
  - See how much work is being done, and where, when and by whom
  - Experiment with a new feature without interfering with working code
  - More?

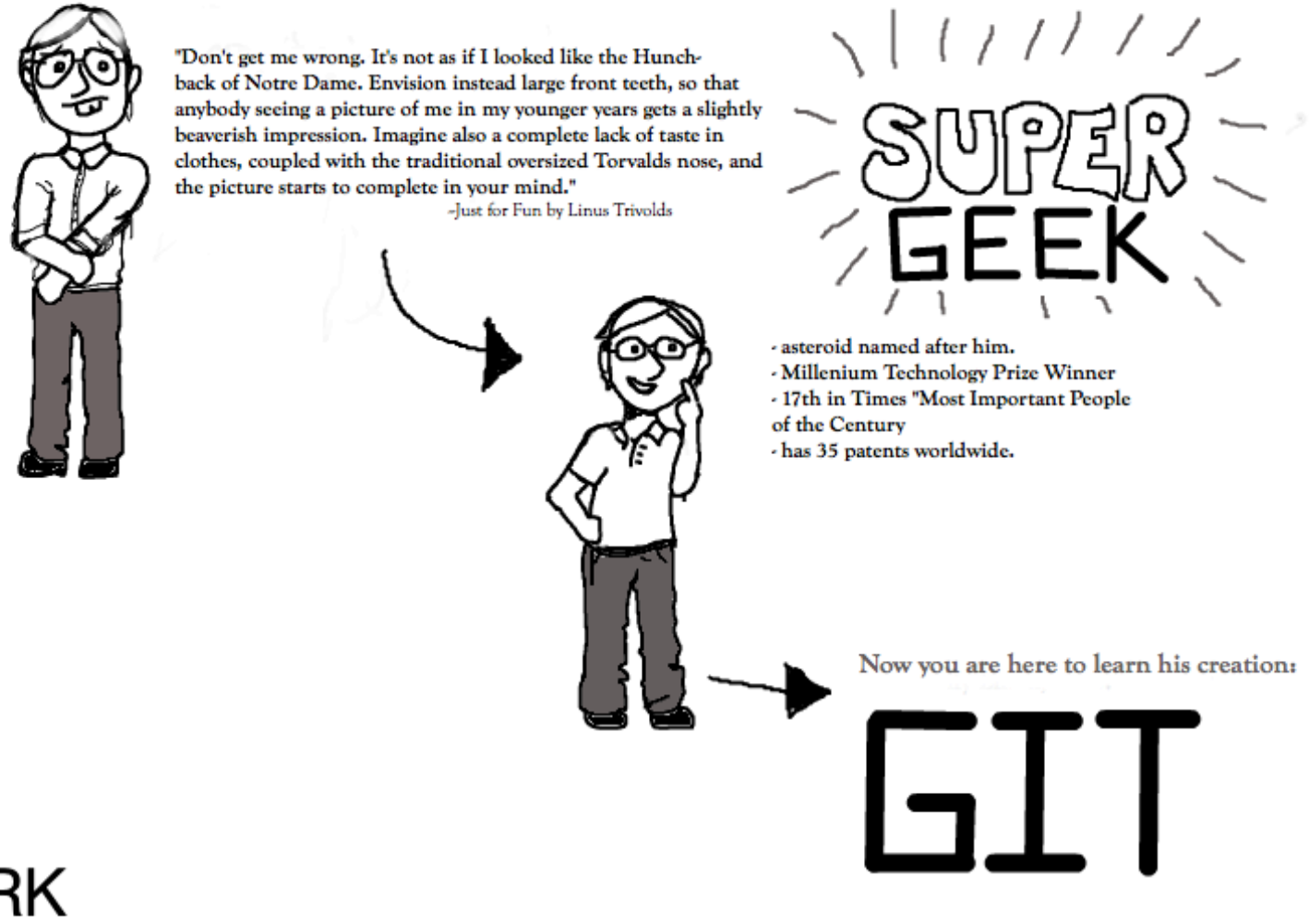
Version control system examples

- Server-based
  - CVS
  - PVCS
  - SourceSafe
  - Subversion
- Distributed
  - Git
  - Mercurial

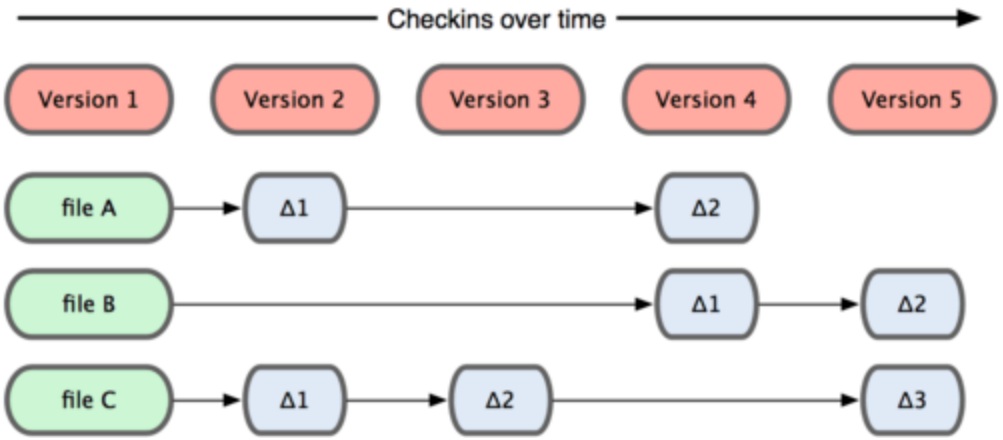
Aside: a little history

- < 2005: Linux using BitKeeper
- 2005: BitKeeper unfriends Linux
- Linus Torvalds and team design Git (uncouth person)
  - Speed
  - Simple design
  - Support for non-linear development
  - Distributed (you can work on the plane)
  - Handle large projects efficiently (speed and data size)

Cartoon – Linus Torvalds

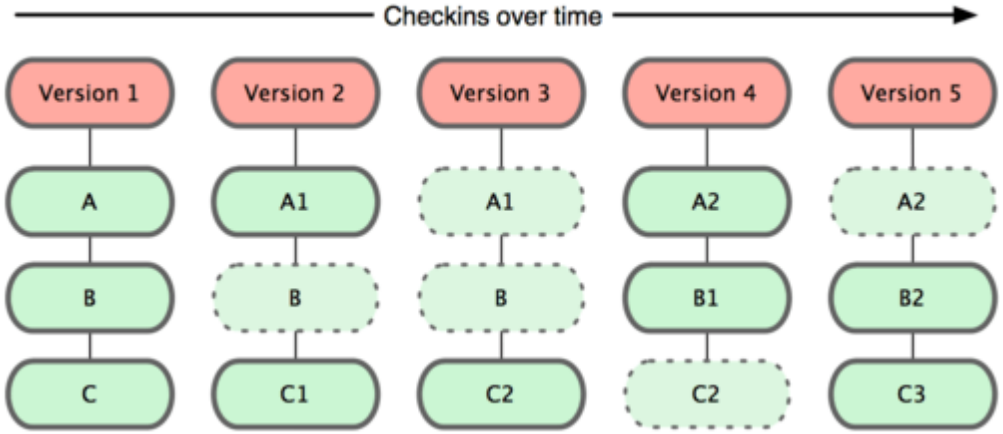


What's the difference?



Other systems

Git



Source: Git book

Terminology

- Key concepts
  - Repository
  - Working Copy
  - Index/Staging area
  - Blobs, Trees
  - Cloning
  - Remotes
  - Pulling + Pushing
  - Local history vs. Public history

How we will approach the terms

- Define
- Give examples
- Hands-on

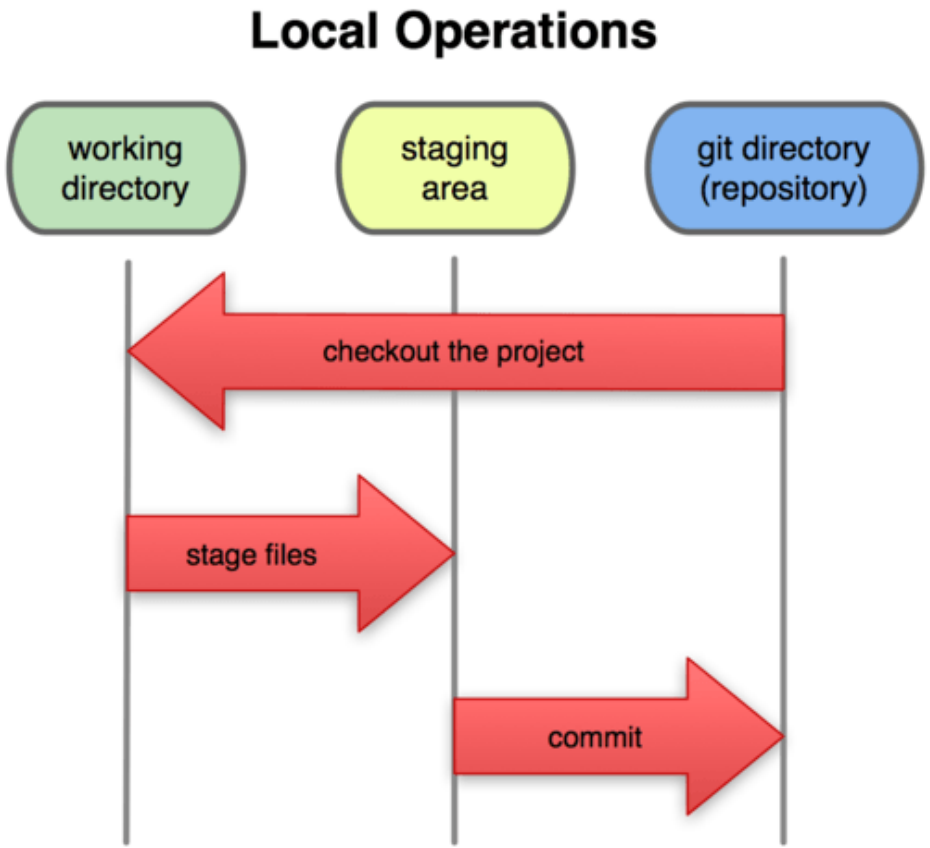
Repository

- A set of files and directories
- Historical record of changes in the repository
- A set of commit objects
- A set of references to commit objects, called heads

- Let us give examples of what qualifies as a repository
  - A copy of a project directory?
  - CVS? Subversion?
- Git is a complete repository, either local and remote

Working copy

- A.k.a "working directory," is a single checkout of one version of the project



Hands-on: analyze the git directory (.git)

Can you have multiple working copies?

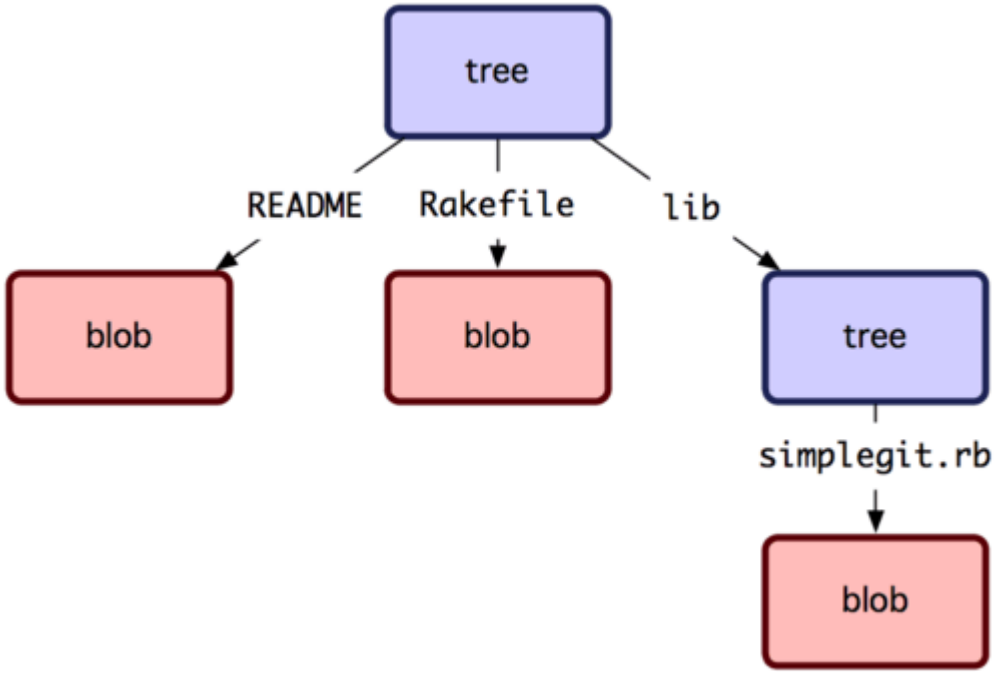
Source: Git book

Index and Staging areas

- Index and Staging area are the same
- It is a simple file in the Git directory
- Stores information about the next commit

Blobs, Trees

- Git is a key-value data store
- You can store a value and get back a key
- There are Git internals
- All we need to know is "tree" and "blob"



Source: Git book

Lab 01 – Install git

- Please do all steps in Lab 01:

- https://github.com/elephantscale/git-labs/tree/main/lab01

Put and get values

Put value, observe the key you get in return

$ echo 'test content' | git hash-object -w –stdin

Find the file:

$ find .git/objects -type f

(SHA-1)

Get it back

git cat-file –p (SHA-1)

Lab 02

Please do all steps in lab 02:

https://github.com/elephantscale/git-labs/tree/main/lab02

Cloning

- Getting a copy of the existing get repository (quick, what is repository?)
- How? `git clone <url>`
- Example:

```
$ git clone git://github.com/schacon/grit.git
```

- Exercises
  - clone on the command line
  - clone in your preferred Git UI (i.e. Eclipse Git, SmartGit, etc.)
  - See the following lab

Lab 03

Please do all steps in lab 03:

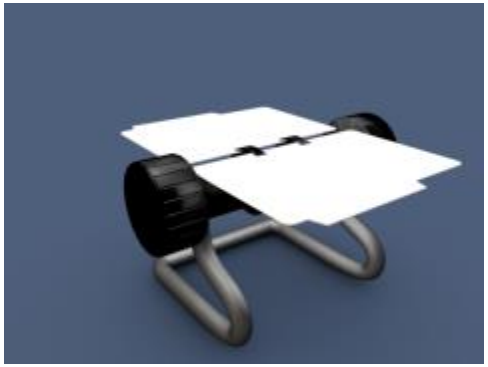
https://github.com/elephantscale/git-labs/tree/main/lab03

Cloning vs 'checkout'

In subversion, this would be **checkout**. Difference?

Git

Subversion



Remotes

- Versions of your project that are hosted on the Internet or network – that's how you collaborate
- Remotes can be
  - Multiple
  - Read only or read-write
  - Try
    - `git remote`

    `origin` – This is where you clone your project from

Pulling + Pushing

- Pulling – from a branch on a remote
- Fetching – all that you don't have yet
- Pushing – back to the branch on a remote

Source: Cutedocpix.com

Lab 04

Please do all steps in lab 04

https://github.com/elephantscale/git-labs/tree/main/lab04

Note, however, that the simples way to work is to always use

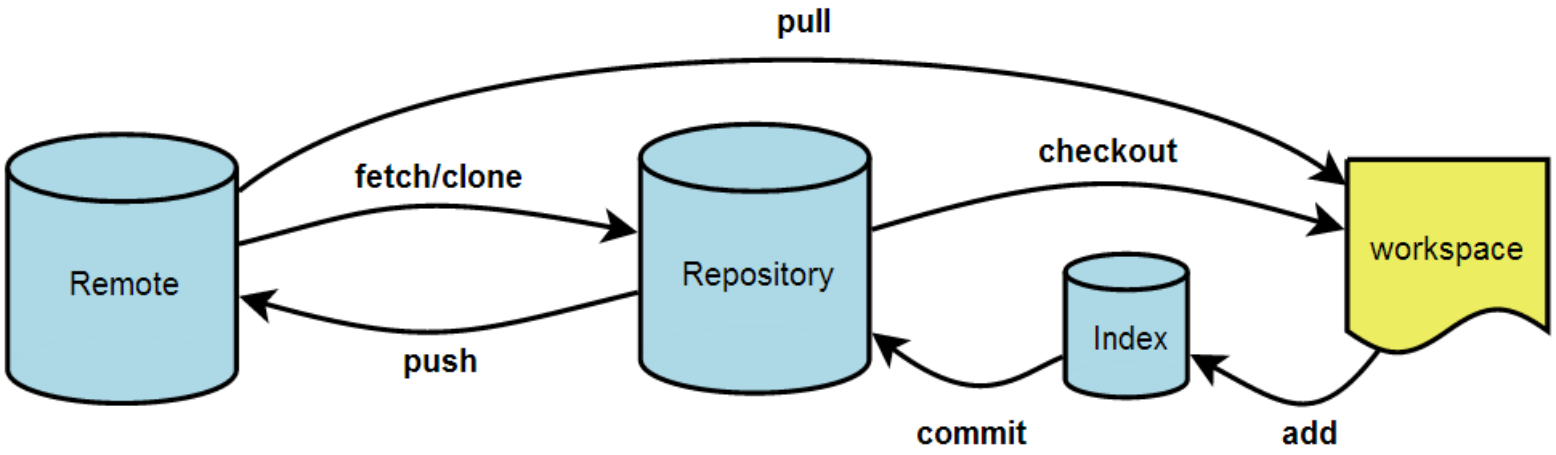
```
git commit -a
```

command

Local history vs. Public history

- Local history is on your laptop
- You can
  - Change commits
  - Change commit messages
  - Reorder
  - Squash
- However
  - Be careful pushing this to the public history
  - Because other developers may end up having to merge

Section – Basic Git Operations



Source: Git book

Viewing a commit in UI

Execute the commands below

In Git Bash

gitk

Or

Tools-Git Shell

gitk

To view a specific commit

git show 5809 (first few letters of the SHA-1)

Switching branches

git checkout <branch-name>

- Try switching between your branches
- Try switching branches to your friend's
- Describe what happens when you switch a branch

Switching branches – practical scenario

- A day in the life of a web developer

Morning

1. Do work on a web site.
2. Create a branch for a new story you're working on.
3. Do some work in that branch.

Let us try that.

1. Work on the text file of your choice
2. Create a branch for a new story your-name_new
3. Do some work in that branch.

Afternoon

- Emergency fix is required in your branch your-name!

1. Switch back to your production branch.
2. Create a branch to add the fix.
3. Test, merge the hotfix branch into your-name, and push to production.
4. Switch back to your original story and continue working.

Let us do that (following lab)

Lab 05

Please do all steps in lab 05

https://github.com/elephantscale/git-labs/tree/main/lab05

Section: making changes, staging, and committing

Source: http://www.northeaststage.com

Making changes, staging and committing – in-depth look

- Staging a commit
- Making a commit
- Pushing your change
- Undoing latest local commit
- Reverting a commit

Staging a commit

- Review: what happens in staging?
- Answer: your changes go to the staging area

Do commands

git status

git add <file>

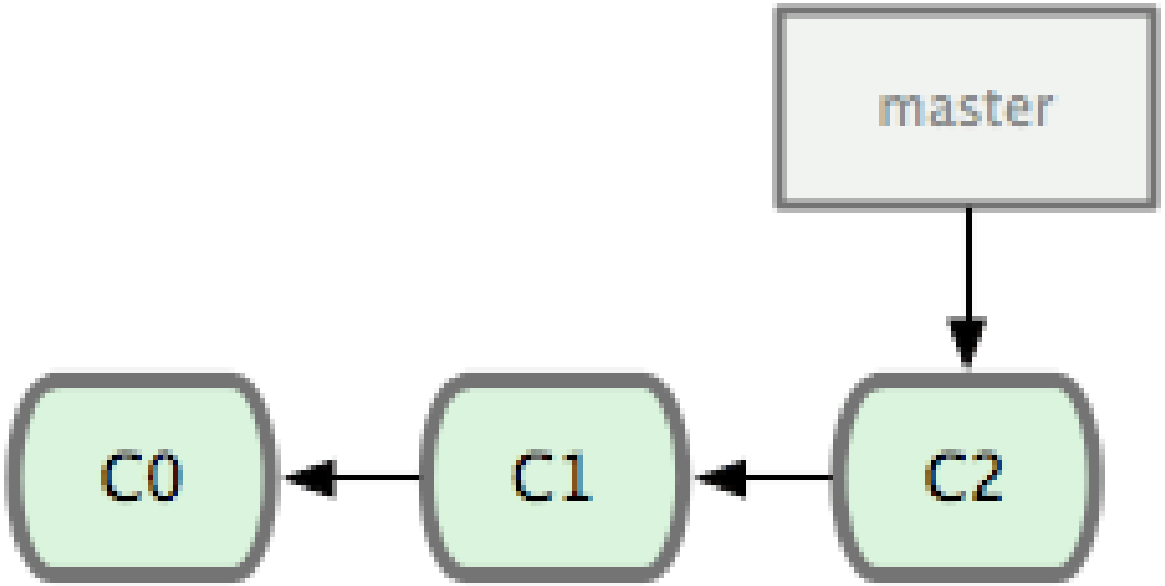
git commit –a

Interactive

git add -i

Making a commit

- Commit is a record of your changes in a Git directory (repository)
- Making a commit is moving the branch point (master in this case) to the next snapshot



Source: Git book

Commits

- What are the differences and similarities between a commitment and a commit?

Commit features

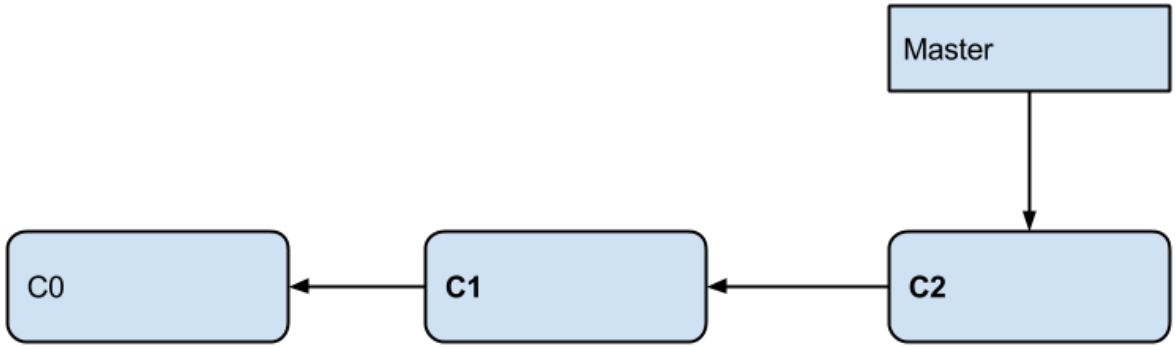
- Permanence
  - Commit leaves a record
  - Commit goes into the Git area
  - Commit can be further recorded in a remote
- Impermanence
  - Commits can be taken back (undone locally or reverted)
  - Commits can be erased (rebase)

How does branching and merging work

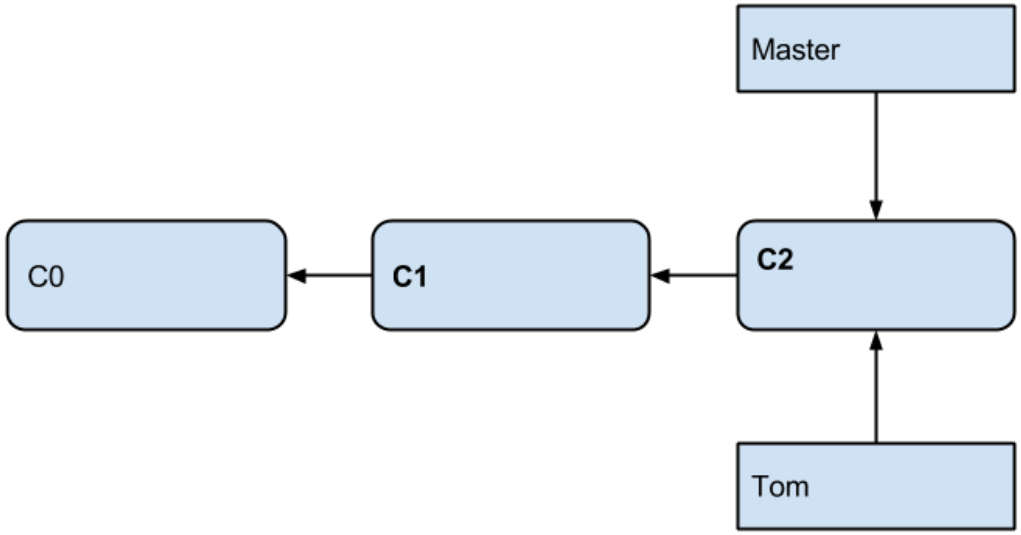
- Let's go back to our morning-afternoon scenario. In brief…

- Working on your issue
- Get interrupted with the production fix
- Fix the production, go back to your issue

This is where you start

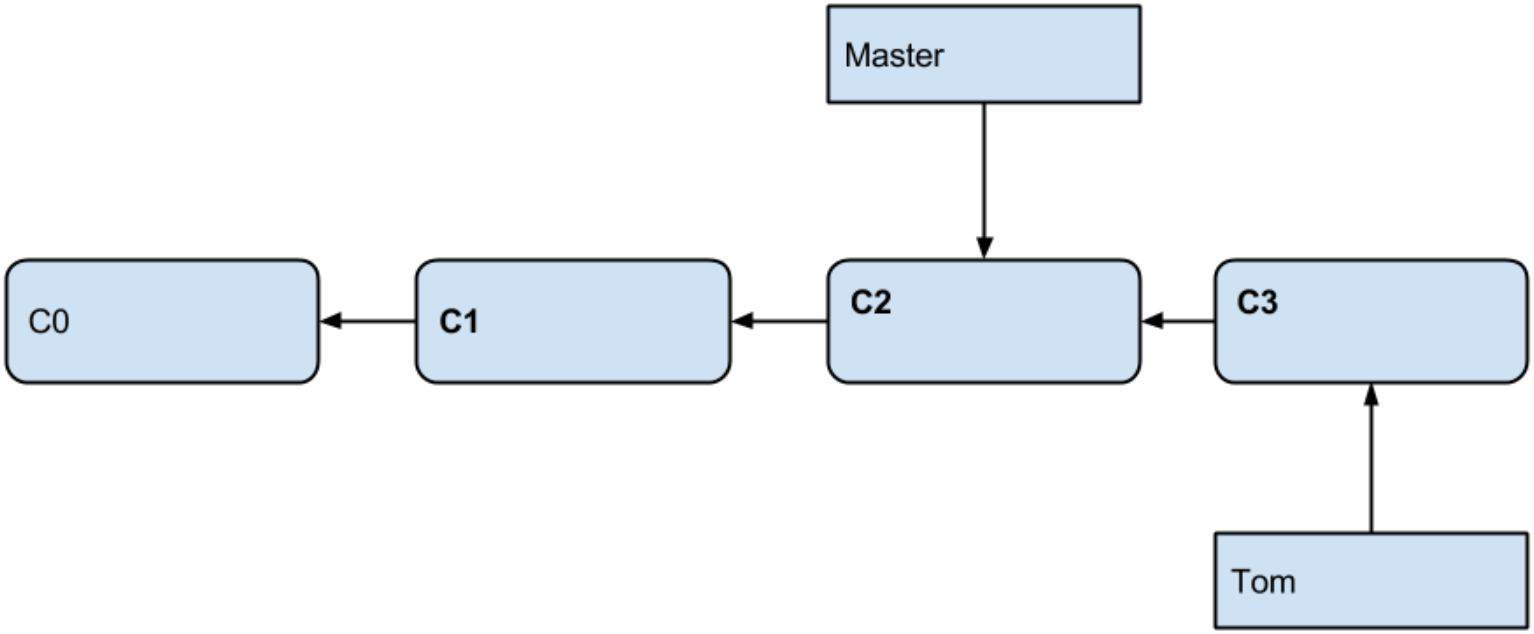


Prepare to work on your feature



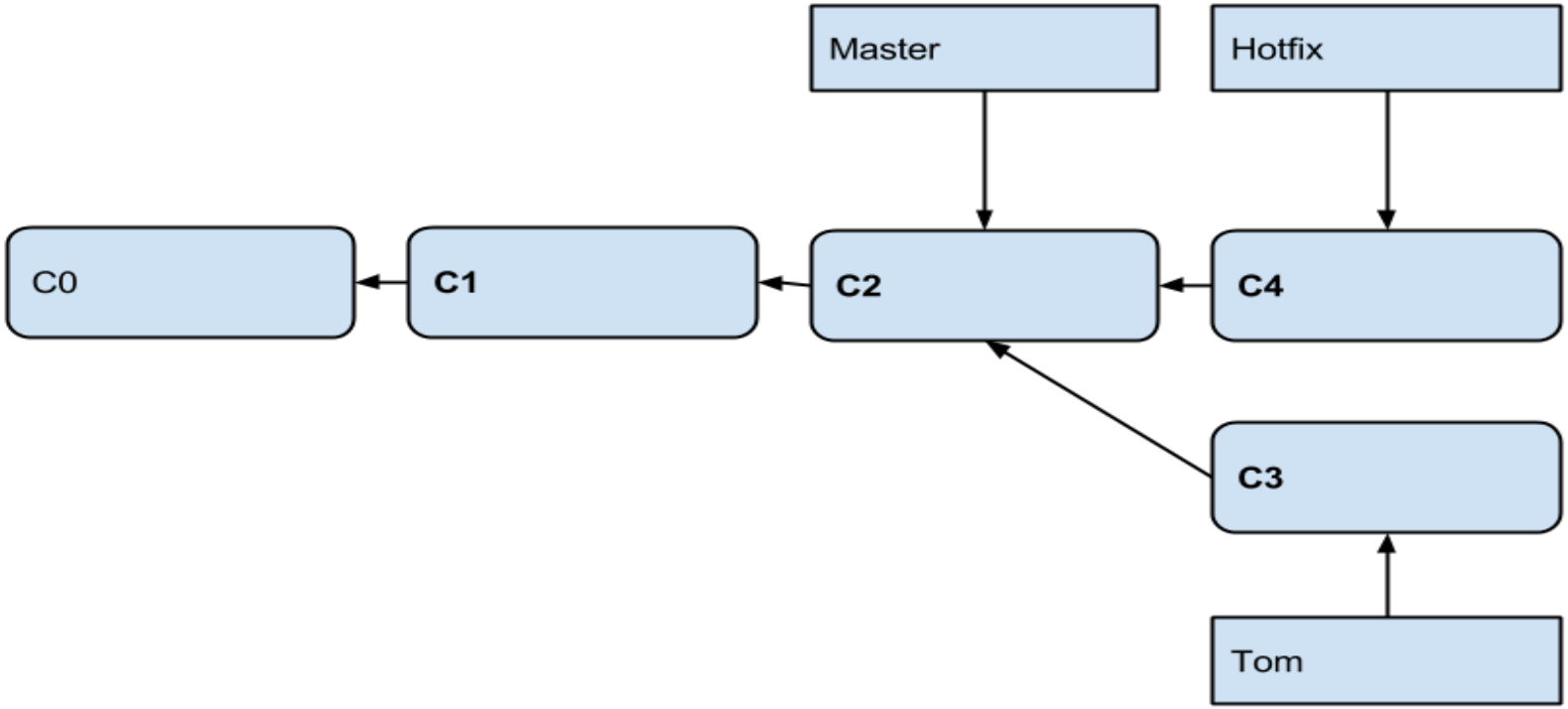
git checkout tom -b

Commit your new changes



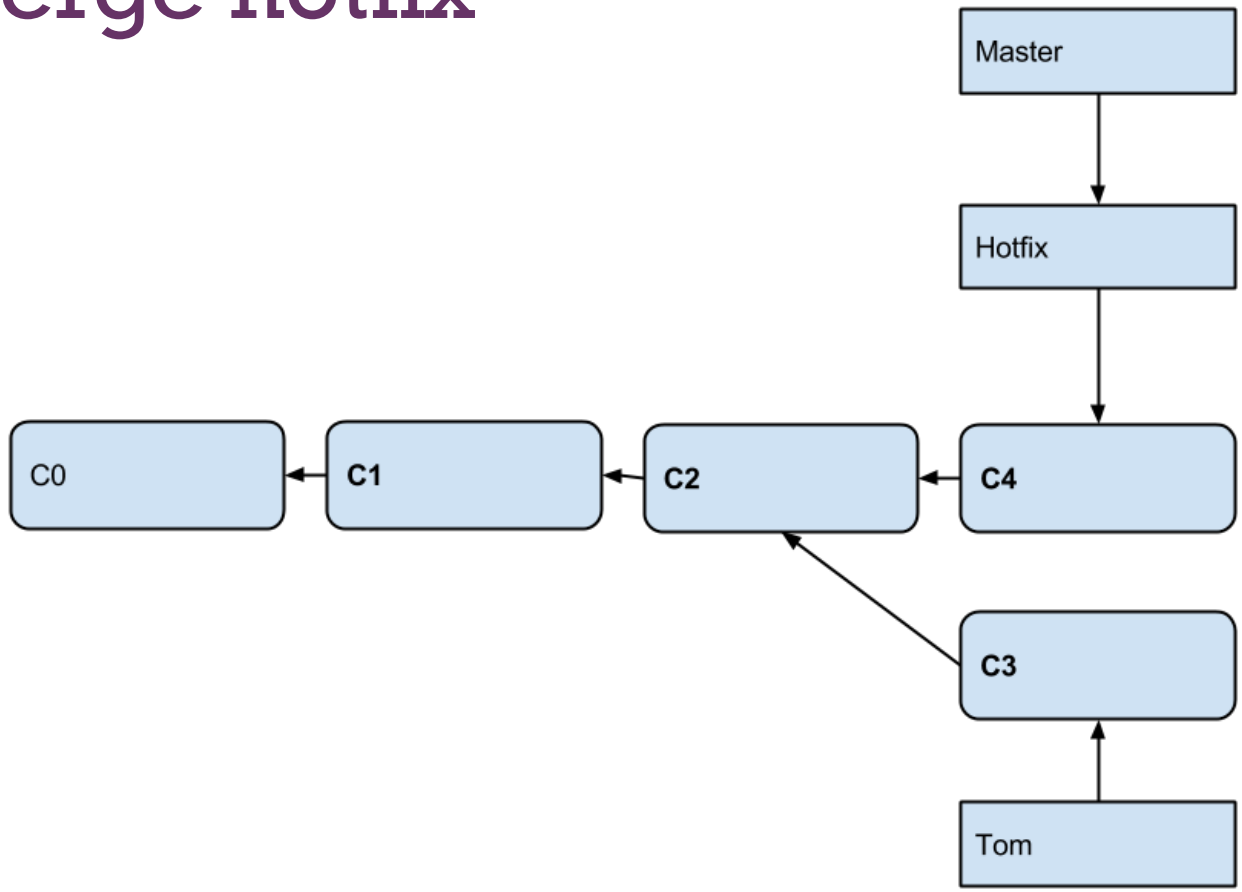
git commit -a

Work on hotfix



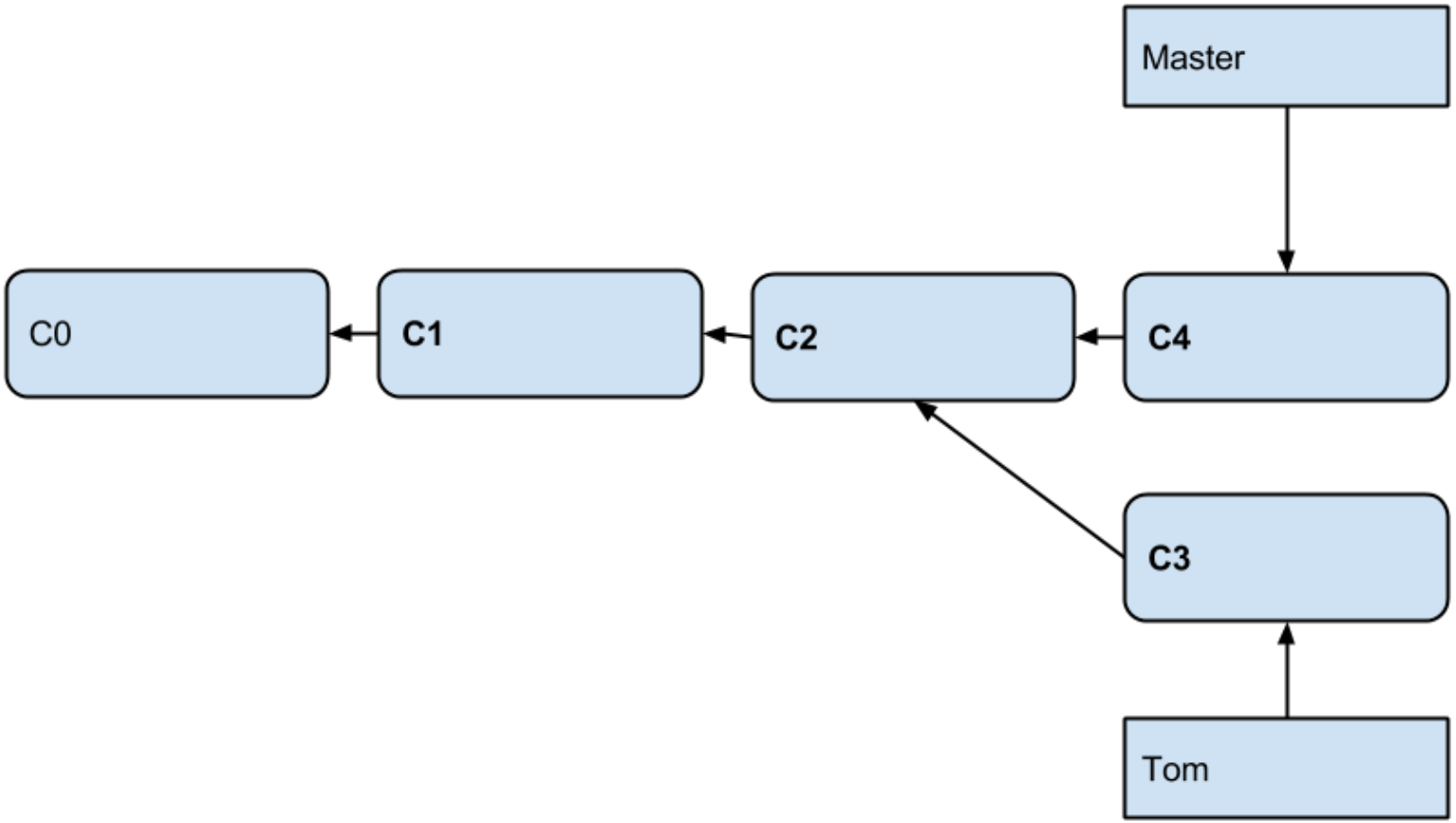

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ do your work
$ git commit -a -m 'urgent fix'
```

Merge hotfix



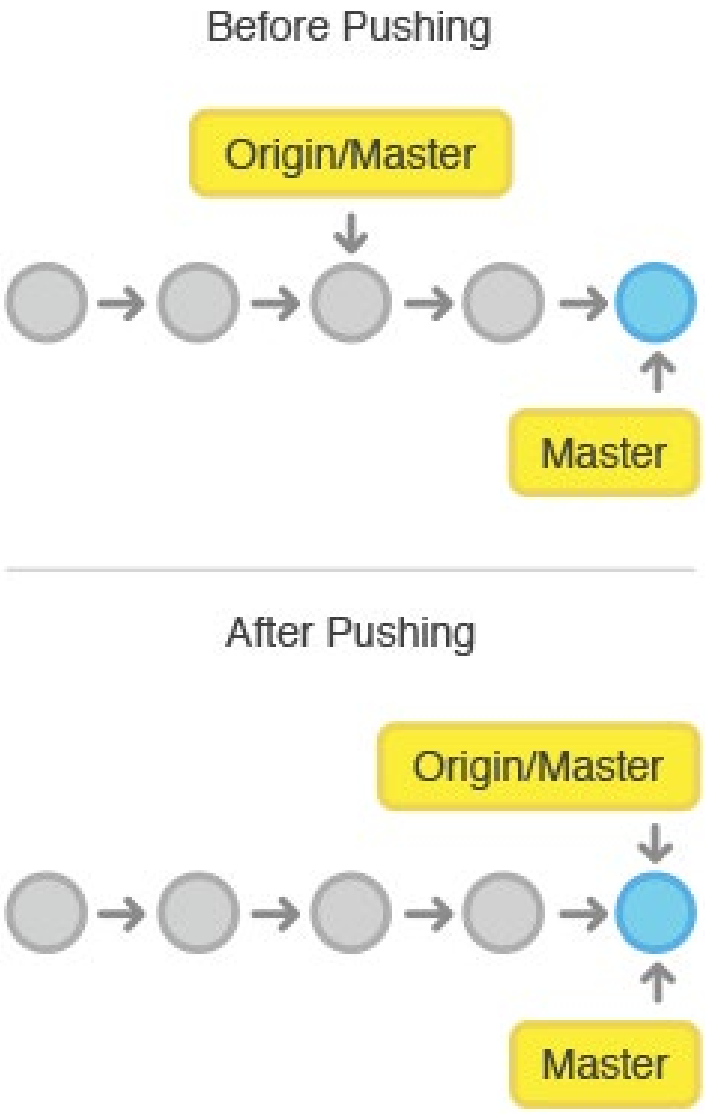
git checkout master
git merge hotfix

And now, clean up!



git branch -d hotfix

Pushing your change



git push <remote> <branch>

or simply

git push

Pushing – do not use force

- As good practice, do not do this

git push <remote> --force

When can it happen?

1) Did not pull but want to push
2) Rebase (we will mention it later)

Undoing latest local commit

Can you do this?

git undo-commit

Yes, if prior you type

```
git config --global alias.undo-commit 'reset --soft HEAD^'
```

Another way:

```
git reset --soft HEAD~1
```

Undo staging

- Say you did this and added too much

```
git add .
```

Here is how you can extricate yourself

```
git rm -r --cached .
```

In the future, you may do 'add' interactively

```
git add -n .
```

Try this: create a file, add it, then undo the staging

Revert commit

To go to a previous commit

```
git checkout 0d1d (start of your hash)
```

Careful! To go back and delete all subsequent commits

```
git reset --hard 0d1d (start of the commit hash)
```

Section: Merge and Conflict resolution

Merge and Conflict resolution

- How merge conflicts happen
- Preventing merge conflicts
- How to resolve a merge conflict

How merge conflicts happen

- Change a file in one branch
- Chart the same file in another branch – same line!
- Now merge one branch into the other

```
git checkout branch1 -b – now edit the file

git checkout branch2 -b – now edit the file

git merge branch1

Auto-merging <your-file>

CONFLICT (content): Merge conflict in <your file>

Automatic merge failed; fix conflicts and then commit
```

How to resolve a merge conflict

1. Git writes markers in the file
2. You edit that file
3. `git add <file>`

Conflict message example

```
git status

# # On branch branch1

# # You have unmerged paths.

# #   (fix conflicts and run "git commit")

# #

# # Unmerged paths:

# #   (use "git add ..." to mark resolution)

# #

# # both modified:      <your file>

# #

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

What you will see

```
Threshold events

<<<<<<< HEAD

two

=======

three

>>>>>>> branch-a
```

Lab 06

Please do all steps in lab 06

https://github.com/elephantscale/git-labs/tree/main/lab06

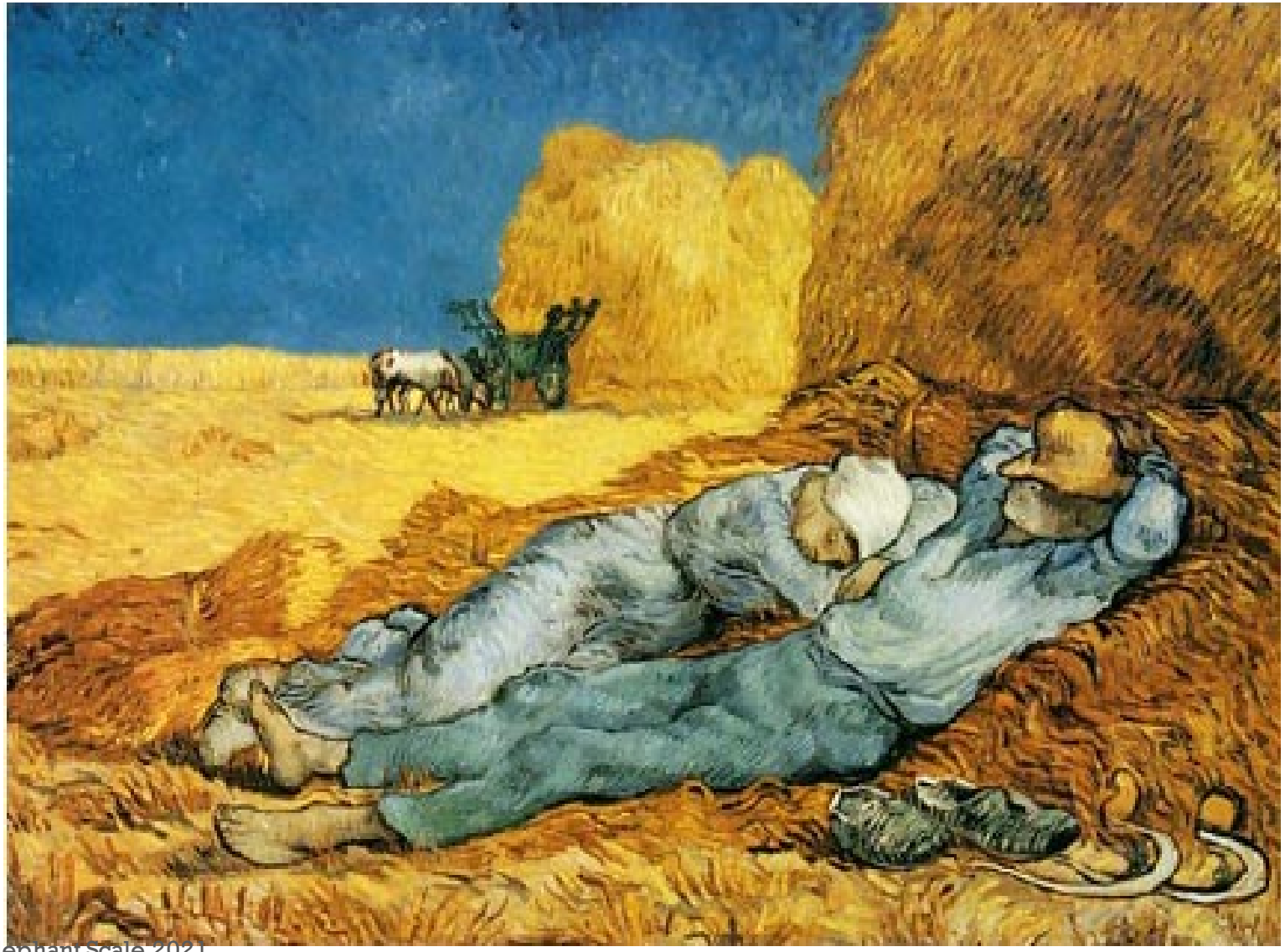
Preventing merge conflicts

- Simple
  - Go with small iterations, in a branch, then merge and delete that branch
  - Do not forget to pull often
- Advanced
  - `git pull rebase`. Git will:
  - Undo (unwind your commits)
  - Pull remote commits
  - Replay your local commits
  - You fix the conflicts if any
  - `git rebase continue`

Tagging

- **tags** - symbolic names for a given *revision*. They always point to the same object (usually: to the same revision); they do not change.
- **branches** - symbolic names for *line of development*. New commits are created on top of branch. The branch pointer naturally advances, pointing to newer and newer commits.

Vincent Van Gogh. Siesta



Detail

Painted late in life
Copied after Millet
Added his own artistic intensity

Pull requests

- Git has no branch security
- Anyone can work in his friend's branch, then commit
- How do you add not-trusted developers to the team?
  - New developers may be given read-only access
  - Then will fork the project but won't be able to commit the changes
  - They then issues a git pull request

Lab 07

Please do all steps in lab 07

https://github.com/elephantscale/git-labs/tree/main/lab07