A minimal Git guide

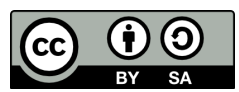
S. Rossi Tisbeni
https://www.unibo.it/sitoweb/simone.rossitisbeni/

Programmazione per la Fisica, Università di Bologna

Anno Accademico 2021/2022

https://virtuale.unibo.it/course/view.php?id=18455

https://baltig.infn.it/giaco/pf2020



What is Git

Git is a tool that protects yourself and others from yourself and others.

What is Git

Git is a tool that protects yourself and others from yourself and others.
Git can:

- Keep record of all the (tracked) files in your directory.

What is Git

Git is a tool that protects yourself and others from yourself and others.
Git can:

- Keep record of all the (tracked) files in your directory.
- Maintain a history of all the changes.

What is Git

Git is a tool that protects yourself and others from yourself and others.
Git can:

- Keep record of all the (tracked) files in your directory.
- Maintain a history of all the changes.
- Revert changes to fix mistakes.

What is Git

Git is a tool that protects yourself and others from yourself and others.
Git can:

- Keep record of all the (tracked) files in your directory.
- Maintain a history of all the changes.
- Revert changes to fix mistakes.
- Allow for concurrent work and help preventing conflicts.

What is Git

Git is a tool that protects yourself and others from yourself and others.
Git can:

- Keep record of all the (tracked) files in your directory.
- Maintain a history of all the changes.
- Revert changes to fix mistakes.
- Allow for concurrent work and help preventing conflicts.

Also known as Version Control

git init

- `git init` initializes the working directory.

git init

- `git init` initializes the working directory.
- It tells Git to start managing a repository for your workspace.

git init

- `git init` initializes the working directory.
- It tells Git to start managing a repository for your workspace.
- The repository (or 'repo' for short) is a folder in a working directory in which Git tracks all changes made to files and builds a history of those changes.

git status

`git status` describes the current state of the repository.

```
~/workspace$ git init↵
```

git status

`git status` describes the current state of the repository.

```
~/workspace$ git init↵
Initialized empty Git repository in ~/workspace/.git/
~/workspace$
```

git status

`git status` describes the current state of the repository.

```
~/workspace$ git init↵
Initialized empty Git repository in ~/workspace/.git/
~/workspace$ git status↵
```

git status

`git status` describes the current state of the repository.

```
~/workspace$ git init↵
Initialized empty Git repository in ~/workspace/.git/
~/workspace$ git status↵
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
~/workspace$
```

git status

`git status` describes the current state of the repository.

```
~/workspace$ git init↵
Initialized empty Git repository in ~/workspace/.git/
~/workspace$ git status↵
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
~/workspace$ rm -r .git↵
~/workspace$
```

git status

`git status` describes the current state of the repository.

```
~/workspace$ git init↵
Initialized empty Git repository in ~/workspace/.git/
~/workspace$ git status↵
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
~/workspace$ rm -r .git↵
~/workspace$ git status↵
```

git status

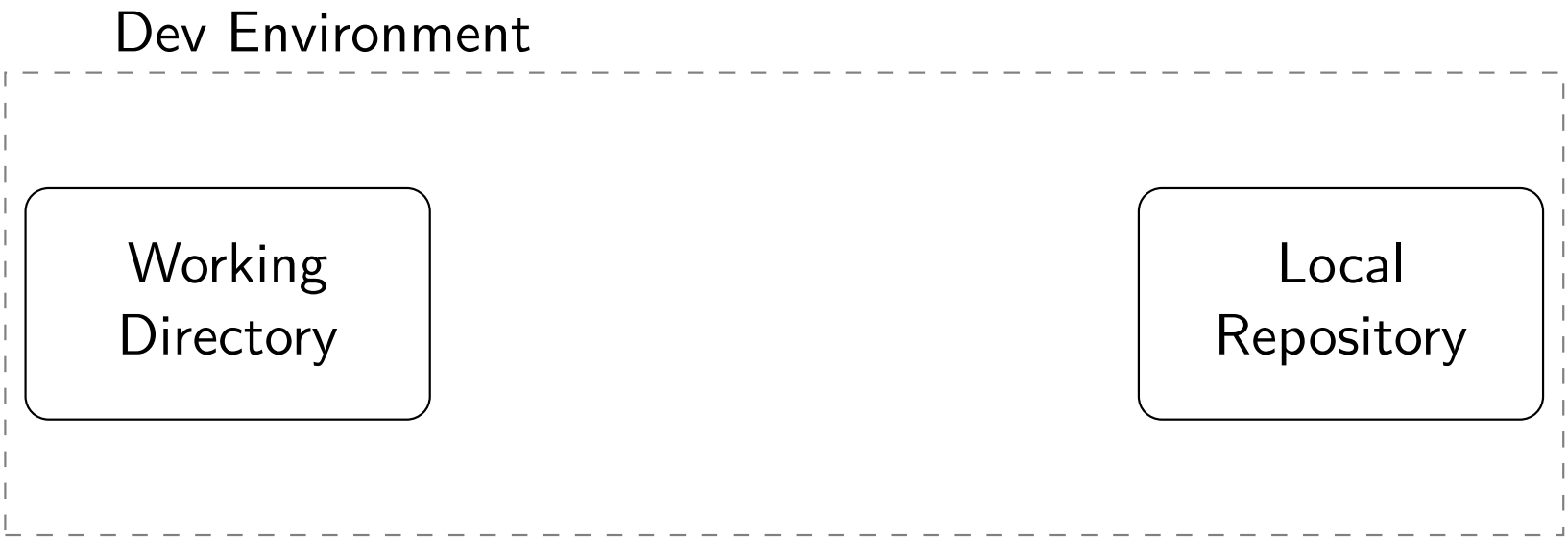
`git status` describes the current state of the repository.

```
~/workspace$ git init↵
Initialized empty Git repository in ~/workspace/.git/
~/workspace$ git status↵
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
~/workspace$ rm -r .git↵
~/workspace$ git status↵
fatal: not a git repository (or any of the parent directories): .git
```

Local Workflow



Untracked files

Files in the working directory are not automatically added to the repository.

```
~/workspace$ touch main.cpp && ls↵
```

Untracked files

Files in the working directory are not automatically added to the repository.

```
~/workspace$ touch main.cpp && ls↵
main.cpp
~/workspace$
```

Untracked files

Files in the working directory are not automatically added to the repository.

```
~/workspace$ touch main.cpp && ls↵
main.cpp
~/workspace$ git status↵
```

Untracked files

Files in the working directory are not automatically added to the repository.

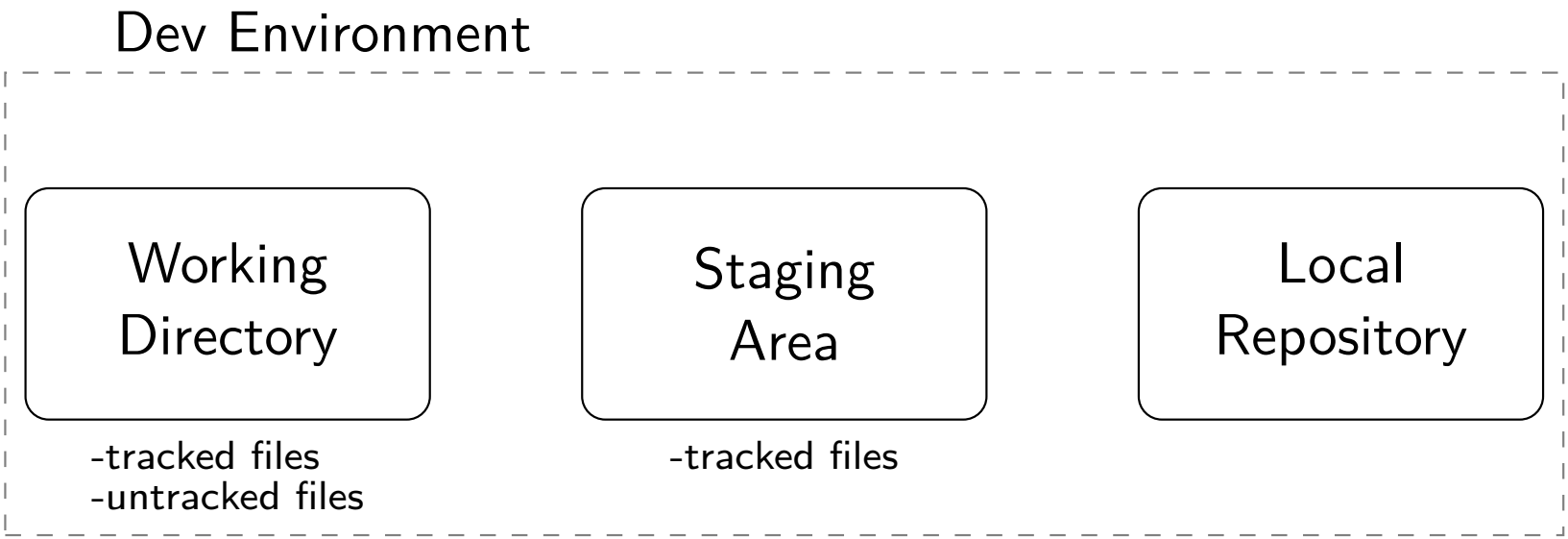
```
~/workspace$ touch main.cpp && ls↵
main.cpp
~/workspace$ git status↵
On branch master

No commits yet

Untracked files:
    (use "git add <file>..." to include in what will be committed)
        main.cpp

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

Local Workflow



git add

- `git add <filename>` asks Git to track changes to a file in the repository.

```
~/workspace$ git add main.cpp↵
```

git add

- `git add <filename>` asks Git to track changes to a file in the repository.

```
~/workspace$ git add main.cpp↵
~/workspace$ git status↵
```

git add

- `git add <filename>` asks Git to track changes to a file in the repository.

```
~/workspace$ git add main.cpp↵
~/workspace$ git status↵
On branch master

No commits yet

Changes to be committed:
    (use "git rm --cached <file>..." to unstage)
        new file:   main.cpp
```

git add (cont.)

- You can add multiple files to the staging area with `git add <file1> <file2> <...>`

git add (cont.)

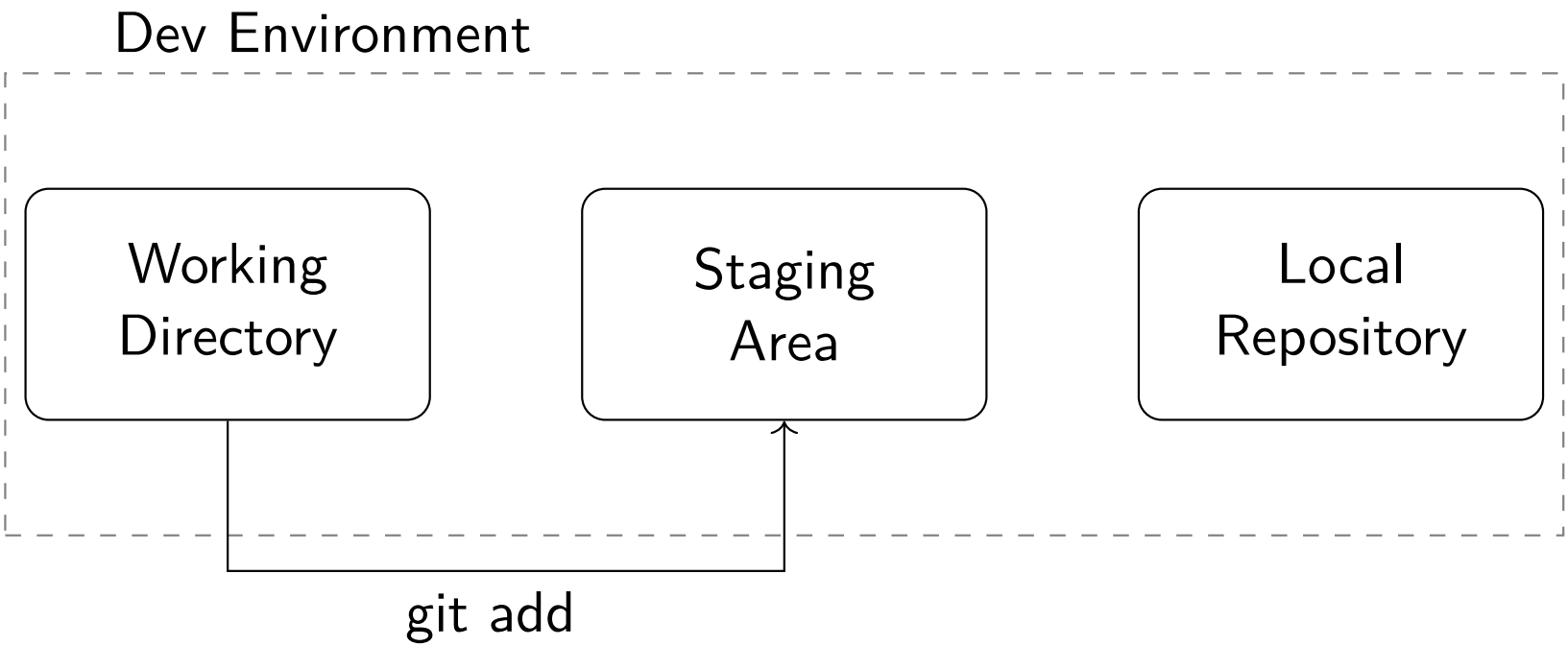
- You can add multiple files to the staging area with `git add <file1> <file2> <...>`
- There is never a good reason to use `git add *`, `git add .` or `git add -A`.

git add (cont.)

- You can add multiple files to the staging area with `git add <file1> <file2> <...>`
- There is never a good reason to use `git add *`, `git add .` or `git add -A`.

Git is used to track changes in text files.
The functionality of Git is reduced for binary files.

Local Workflow



git commit -m

- Files in the staging area are tracked by Git, but their changes have not been saved into the repository.
- With `git commit` a collection of changes in the staging area is checked into the local repository and inserted into the repo history.
- Each commit should be labelled with a message that clearly describes the changes made.
- The commit message can be specified between quotation marks:

```
git commit -m "Commit message"
```

git commit -m (cont.)

- `git commit -m "Commit message"` asks Git to save changes into the local repository history.

```
~/workspace$ git commit -m "Initial commit"↵
```

git commit -m (cont.)

- `git commit -m "Commit message"` asks Git to save changes into the local repository history.

```
~/workspace$ git commit -m "Initial commit"↵
[master (root-commit) 0367896] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 main.cpp
/workspace$ git status↵
```

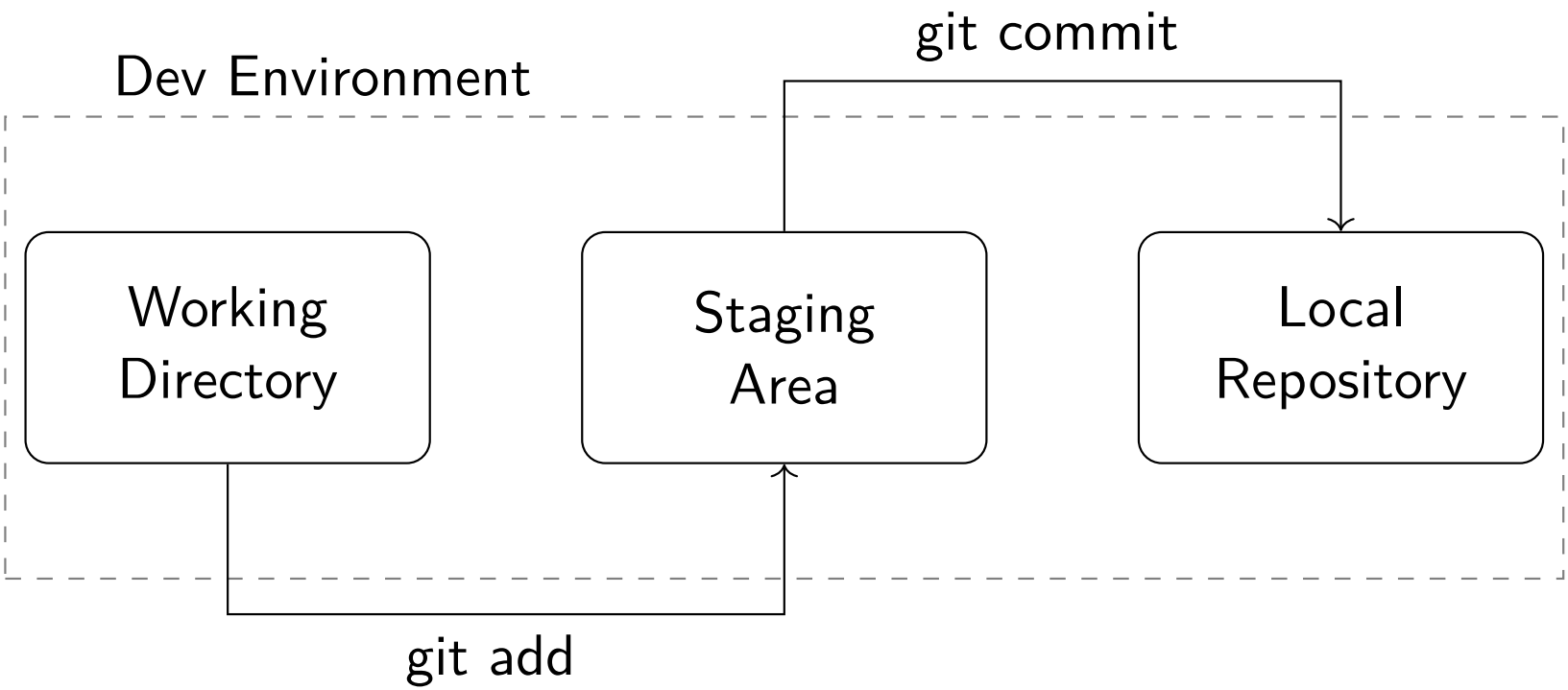
git commit -m (cont.)

- `git commit -m "Commit message"` asks Git to save changes into the local repository history.

```
~/workspace$ git commit -m "Initial commit"↵
[master (root-commit) 0367896] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 main.cpp
/workspace$ git status↵
On branch master
nothing to commit, working tree clean
```

If this is your first time working with Git, you will be prompted to enter your email and name, with which to sign your commits. Follow the instructions, then redo the commit.

Local Workflow



git log

- `git log` shows the local repository history.

```
/workspace$ git log↵
commit 03678964f710e1ef1e9d6c0704e3f02f014109a0 (HEAD -> master)
Author: Simone Rossi Tisbeni <simone.rossitisbeni@unibo.it>
Date:   Fri Jan 29 14:39:12 2021 +0000

    Initial commit
```

git log

- `git log` shows the local repository history.

```
/workspace$ git log↵
commit 03678964f710e1ef1e9d6c0704e3f02f014109a0 (HEAD -> master)
Author: Simone Rossi Tisbeni <simone.rossitisbeni@unibo.it>
Date:   Fri Jan 29 14:39:12 2021 +0000

    Initial commit
```

- Each commit has its own unique identifier.

git diff

- `git diff` is used to show the changes in the files.
- When no argument is passed, it shows any uncommitted changes since the last commit.
- When `--staged` is passed as argument, it shows the difference between the last commit and the staged changes.
- When a commit id is passed as argument, it shows the difference between the last commit and the selected commit.

Branching

- A Git repository keeps track of the history of changes.

Branching

- A Git repository keeps track of the history of changes.
- It can also keep track of multiple lines of development.

Branching

- A Git repository keeps track of the history of changes.
- It can also keep track of multiple lines of development.
- These are known as branches.

git branch

- `git branch` is the command used to manipulate different development lines.

git branch

- `git branch` is the command used to manipulate different development lines.
- When launched with no arguments, it lists the branches available in the repository.

git branch

- `git branch` is the command used to manipulate different development lines.
- When launched with no arguments, it lists the branches available in the repository.

```
~/workspace$ git branch↵
* master
```

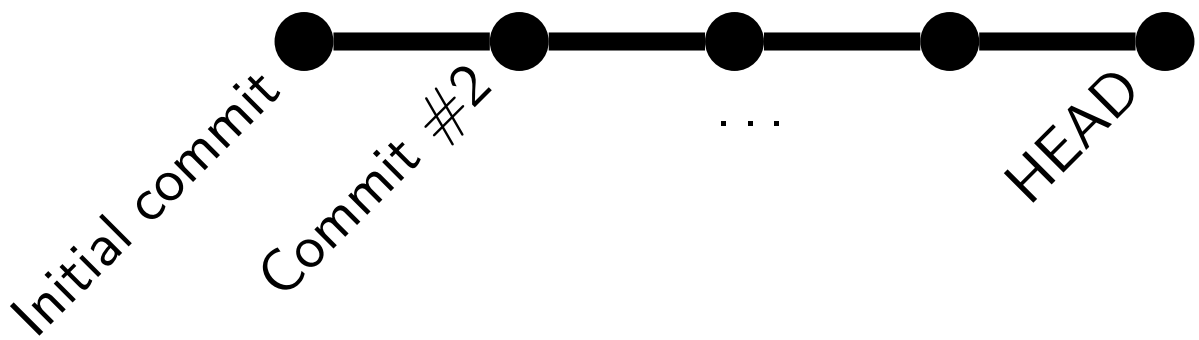
git branch

- `git branch` is the command used to manipulate different development lines.
- When launched with no arguments, it lists the branches available in the repository.

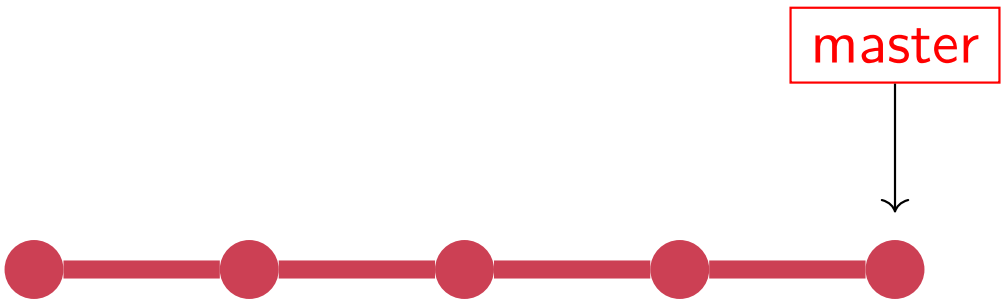
```
~/workspace$ git branch↵
* master
```

- The current branch is highlighted in green and prefixed with * (asterisk).

History Graph



History Graph



Main branch

- The default branch created at the initialization of the repository is usually named master.

Main branch

- The default branch created at the initialization of the repository is usually named master.
- The default branch should host the latest stable version of the repository.

Main branch

- The default branch created at the initialization of the repository is usually named master.
- The default branch should host the latest stable version of the repository.
- The deafult is slowly moving towards a more inclusive denomination.

Main branch

- The default branch created at the initialization of the repository is usually named master.
- The default branch should host the latest stable version of the repository.
- The deafult is slowly moving towards a more inclusive denomination.
- Github already started using (Oct 2020) the name main.

```
~/workspace$ git branch -m master main↵
~/workspace$ git branch↵
* main
```

git branch (cont.)

- Additional branches are used to host feature development, bug fixes and beta releases.
- They are meant to have a very brief life.
- You can create a new branch using `git branch <branch_name>`

git branch (cont.)

- Additional branches are used to host feature development, bug fixes and beta releases.
- They are meant to have a very brief life.
- You can create a new branch using `git branch <branch_name>`

```
~/workspace$ git branch new_branch↵
~/workspace$ git branch↵
* main
  new_branch
```

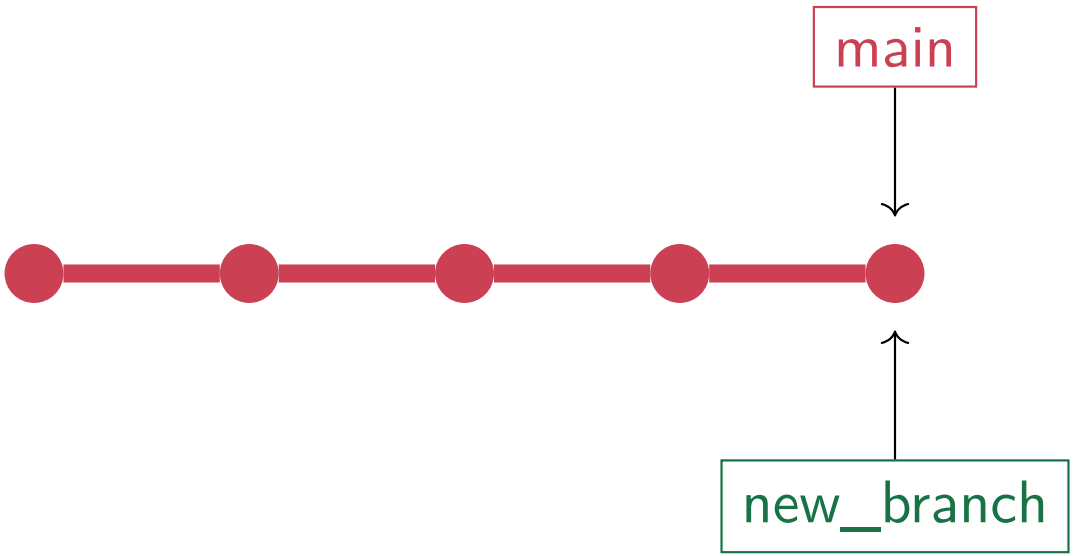
git branch (cont.)

- Additional branches are used to host feature development, bug fixes and beta releases.
- They are meant to have a very brief life.
- You can create a new branch using `git branch <branch_name>`

```
~/workspace$ git branch new_branch↵
~/workspace$ git branch↵
* main
  new_branch
```

- `git log` shows the position of the branches on the commit history.

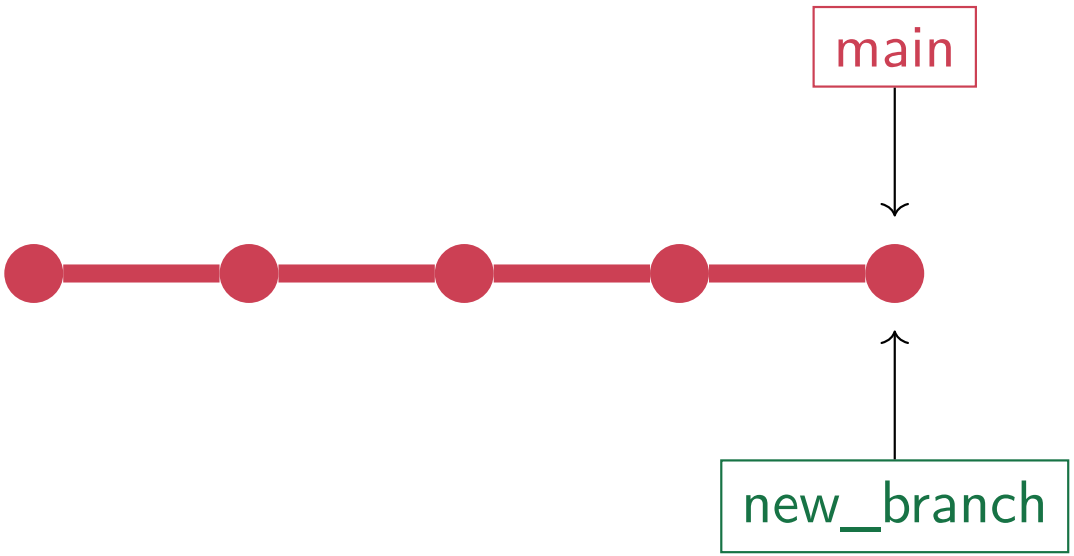
History Graph



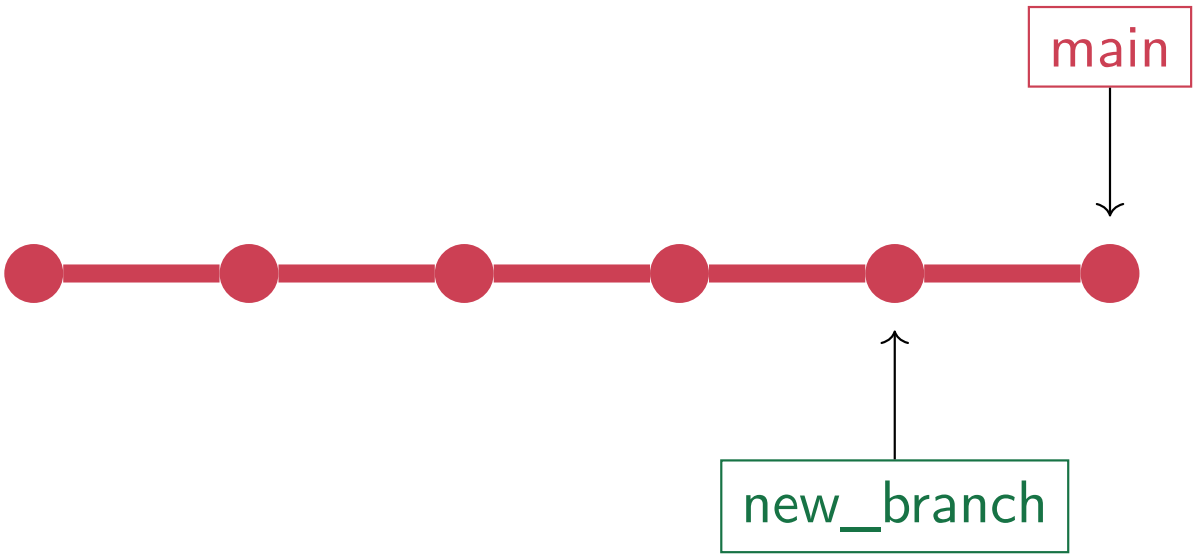
git checkout

- The commit histories are independent on different branches.

git checkout



git checkout



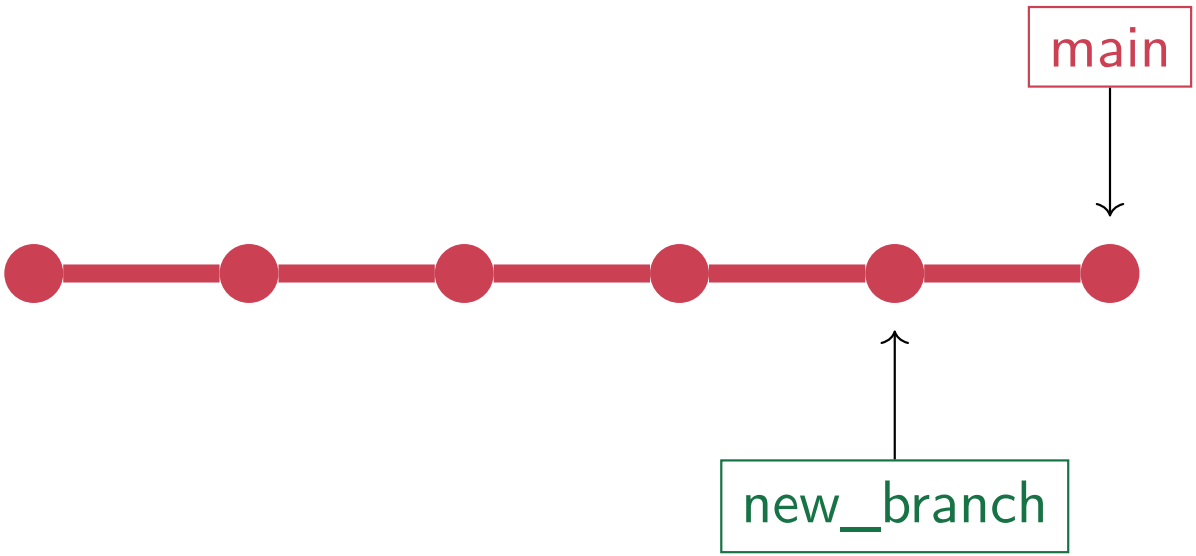
git checkout

- The commit histories are independent on different branches.
- Any new commit will be available only on the active branch.

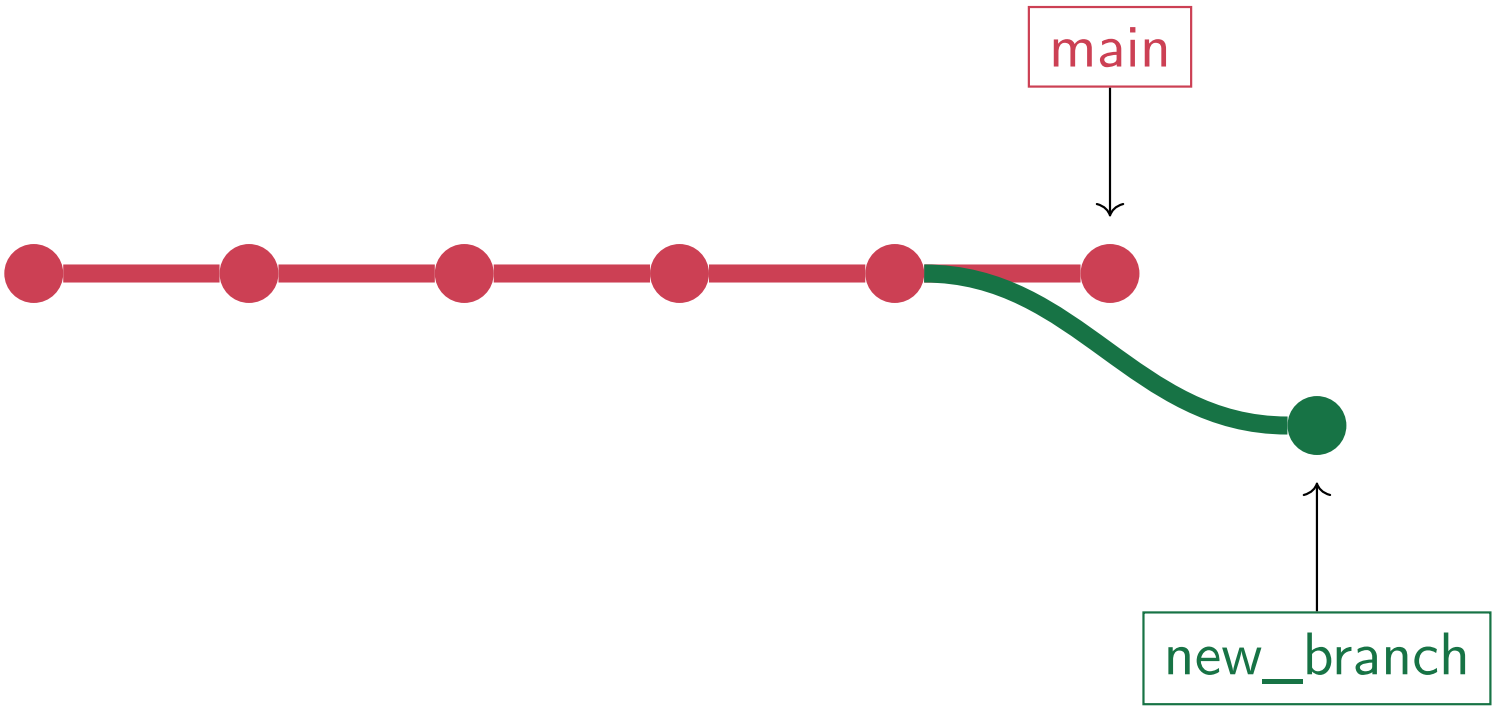
git checkout

- The commit histories are independent on different branches.
- Any new commit will be available only on the active branch.
- To switch branch, you use `git checkout <branch_name>`

git checkout



git checkout



Merge

- Merging is Git way of putting a forked history back together.

Merge

- Merging is Git way of putting a forked history back together.
- It is often used to merge a feature developed in a secondary branch to the main branch.

Merge

- Merging is Git way of putting a forked history back together.
- It is often used to merge a feature developed in a secondary branch to the main branch.
- Git can automatically merge commits unless there are changes that conflict in both commit sequences.

Merge (cont.)

- To merge two branches togheter, first checkout to the destination branch

Merge (cont.)

- To merge two branches togheter, first checkout to the destination branch
- Then use `git merge <branch_name>`

Merge (cont.)

- To merge two branches togheter, first checkout to the destination branch
- Then use `git merge <branch_name>`

```
/workspace$ git checkout main ↵
/workspace$ git merge new_branch
```

Merge (cont.)

- To merge two branches togheter, first checkout to the destination branch
- Then use `git merge <branch_name>`

```
/workspace$ git checkout main ↵
/workspace$ git merge new_branch ↵
/workspace$
```

Merge (cont.)

- To merge two branches togheter, first checkout to the destination branch
- Then use `git merge <branch_name>`

```
 /workspace$ git checkout main ↵
 /workspace$ git merge new_branch ↵
              Auto-merging main.cpp
CONFLICT (content): Merge conflict in main.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

Merge conflicts

- If both the merging branches changed the same lines of the same file, Git won't be able to figure out which version to use.

Merge conflicts

- If both the merging branches changed the same lines of the same file, Git won't be able to figure out which version to use.
- Running `git status` shows which files have conflicts that need to be resolved.

Merge conflicts

- If both the merging branches changed the same lines of the same file, Git won't be able to figure out which version to use.
- Running `git status` shows which files have conflicts that need to be resolved.

```
~/workspace$ git status ↵
On branch main
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.cpp

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

Merge conflicts (cont.)

- Git modifies the content of the affected files with visual indicators.

Merge conflicts (cont.)

- Git modifies the content of the affected files with visual indicators.
- These mark the conflicting section in the receiving branch and the upcoming changes.

Merge conflicts (cont.)

- Git modifies the content of the affected files with visual indicators.
- These mark the conflicting section in the receiving branch and the upcoming changes.

```
This content is not affected by the conflict
<<<<<<< main
this is conflicting text from main
=======
this is conflicting text from the new branch
>>>>>>> new_branch;
```

Merge conflicts (cont.)

- To fix the conflict, simply edit the files with the changes to maintain.

Merge conflicts (cont.)

- To fix the conflict, simply edit the files with the changes to maintain.
- Git doesn't care for the content of the files, it simply requires you to remove the markers.

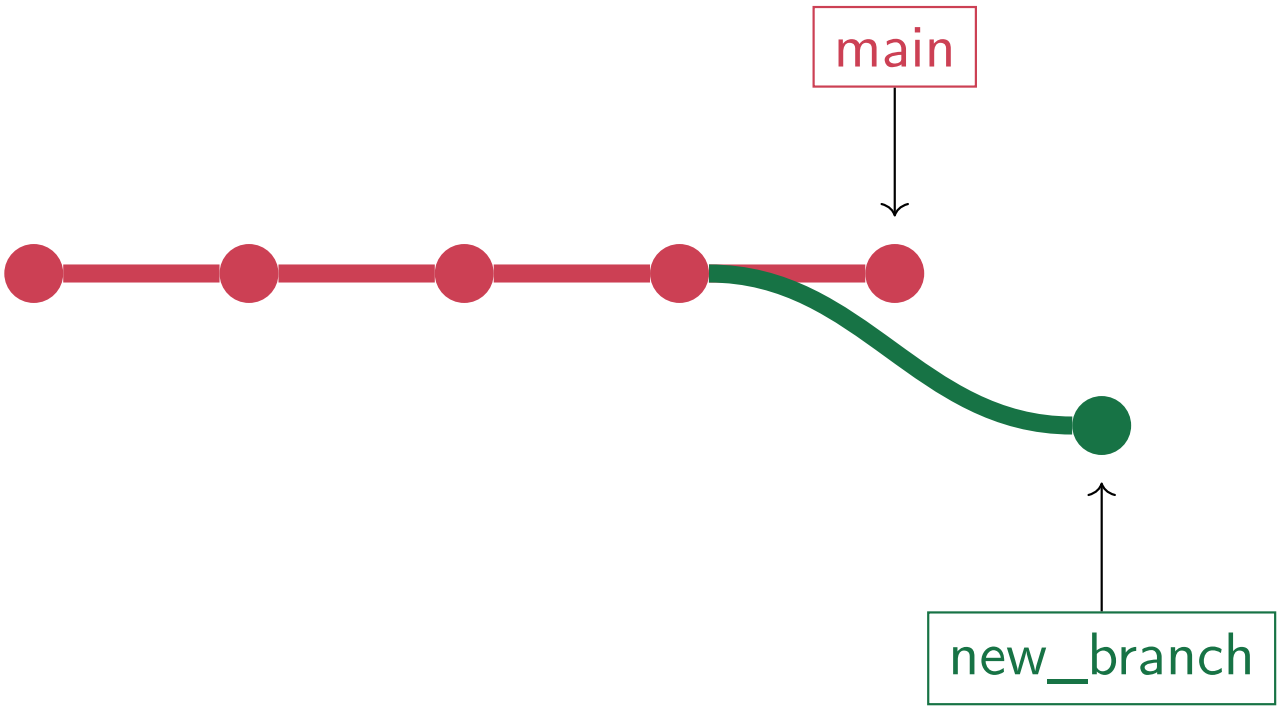
Merge conflicts (cont.)

- To fix the conflict, simply edit the files with the changes to maintain.
- Git doesn't care for the content of the files, it simply requires you to remove the markers.
- Run `git add` on the files to tell Git they are resolved.

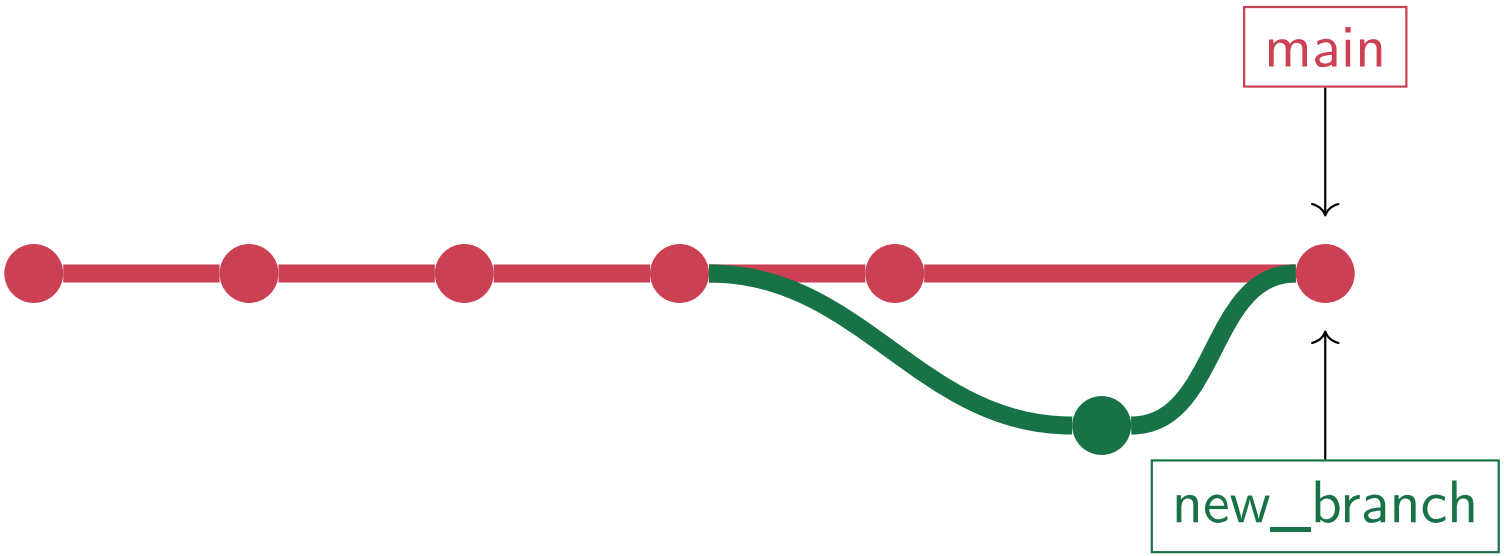
Merge conflicts (cont.)

- To fix the conflict, simply edit the files with the changes to maintain.
- Git doesn't care for the content of the files, it simply requires you to remove the markers.
- Run `git add` on the files to tell Git they are resolved.
- Finally commit the changes with `git commit`.

History Graph



History Graph



git commit

- When running `git commit` with no arguments, Git will open the default text editor.

git commit

- When running `git commit` with no arguments, Git will open the default text editor.
- This will allow you to create a more complex commit message.

git commit

- When running `git commit` with no arguments, Git will open the default text editor.
- This will allow you to create a more complex commit message.

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. The blank line separating the summary from the body
is critical. Further paragraphs come after blank lines.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how.

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.
- This editor does not show a UI or command shortcut.

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.
- This editor does not show a UI or command shortcut.
- You can enter `INSERT` mode by hitting `I` and then editing the text.

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.
- This editor does not show a UI or command shortcut.
- You can enter `INSERT` mode by hitting `I` and then editing the text.
- Press `ESC` to exit `INSERT` mode, followed by `:wq` to save changes and quit.

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.
- This editor does not show a UI or command shortcut.
- You can enter `INSERT` mode by hitting `I` and then editing the text.
- Press `ESC` to exit `INSERT` mode, followed by `:wq` to save changes and quit.
- The default on WSL is Nano, a more user-friendly alternative.

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.
- This editor does not show a UI or command shortcut.
- You can enter `INSERT` mode by hitting `I` and then editing the text.
- Press `ESC` to exit `INSERT` mode, followed by `:wq` to save changes and quit.
- The default on WSL is Nano, a more user-friendly alternative.
- This editor shows a list of common shortcuts at the bottom of the Terminal.

Default text editor

Git will load the commit message on the default text editor.

- Vim is the default editor for many Linux distribution and for Git bash on Windows.
- This editor does not show a UI or command shortcut.
- You can enter `INSERT` mode by hitting `I` and then editing the text.
- Press `ESC` to exit `INSERT` mode, followed by `:wq` to save changes and quit.
- The default on WSL is Nano, a more user-friendly alternative.
- This editor shows a list of common shortcuts at the bottom of the Terminal.
- Directly edit your file, then hit `Ctrl+O` to save and `Ctrl+X` to quit.

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`
- You can change the default text editor for Git with `git config --global core.editor nano`

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`
- You can change the default text editor for Git with `git config --global core.editor nano`
- You can also have nano be the default text editor for the OS.

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`
- You can change the default text editor for Git with `git config --global core.editor nano`
- You can also have nano be the default text editor for the OS.
- Use `nano ~/.bashrc` to edit the startup script.

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`
- You can change the default text editor for Git with `git config --global core.editor nano`
- You can also have nano be the default text editor for the OS.
- Use `nano ~/.bashrc` to edit the startup script.
- Add the following lines

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`
- You can change the default text editor for Git with `git config --global core.editor nano`
- You can also have nano be the default text editor for the OS.
- Use `nano ~/.bashrc` to edit the startup script.
- Add the following lines

```
export VISUAL=nano
export EDITOR=nano
```

Default text editor (cont.)

- If nano is not installed use `sudo apt install nano`
- You can change the default text editor for Git with `git config --global core.editor nano`
- You can also have nano be the default text editor for the OS.
- Use `nano ~/.bashrc` to edit the startup script.
- Add the following lines

```
export VISUAL=nano
export EDITOR=nano
```

- Save and quit with `Ctrl+O` `Ctrl+X`

.gitignore

- Git should be used only for non-generated files.

.gitignore

- Git should be used only for non-generated files.
- The output of a compilation (for example) should not be tracked by the repository.

.gitignore

- Git should be used only for non-generated files.
- The output of a compilation (for example) should not be tracked by the repository.
- Git will automatically ignore anything listed in a file called `.gitignore`

.gitignore

- Git should be used only for non-generated files.
- The output of a compilation (for example) should not be tracked by the repository.
- Git will automatically ignore anything listed in a file called `.gitignore`

```
# Lines starting with # are comments
# Blank lines can be used as separator for readability

# The pattern below match a specific file which will be excluded
exlcuded-file

# The pattern below match a folder and all its content
build/

# The pattern below matches any file with extension .out
*.out
```

references

Git tutorials and guides:

- https://www.atlassian.com/git/tutorials
- https://hsf-training.github.io/analysis-essentials/git/README.html

Git cheatsheet

- http://ndpsoftware.com/git-cheatsheet.html

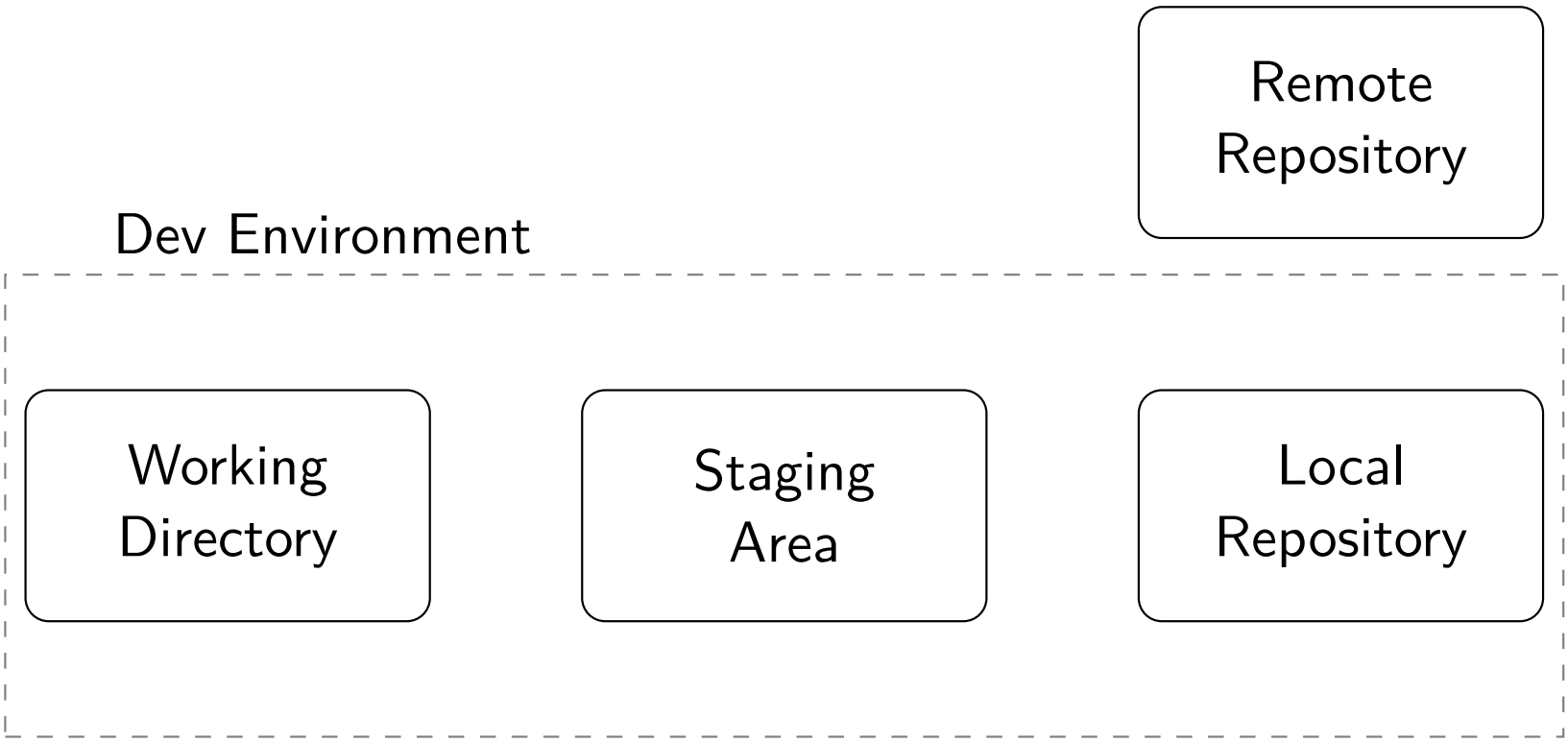
How to write a commit message

- https://chris.beams.io/posts/git-commit/

If you are interested in Vim

- https://www.openvim.com/

Remote Workflow



Why remote?

Working on a remote repository allows to:

- Maintain an online backup of your work and the entire history of changes.

Why remote?

Working on a remote repository allows to:

- Maintain an online backup of your work and the entire history of changes.
- Collaborate with multiple users on the same files.

Why remote?

Working on a remote repository allows to:

- Maintain an online backup of your work and the entire history of changes.
- Collaborate with multiple users on the same files.
- Share your work with a community or publicly.

Github

- Github is a provider for hosting repositories online for free.

Github

- Github is a provider for hosting repositories online for free.
- It allows secure access to private and public repositories

Github

- Github is a provider for hosting repositories online for free.
- It allows secure access to private and public repositories
- To host your repository on Github, first create an account at `https://github.com/join`

Github

- Github is a provider for hosting repositories online for free.
- It allows secure access to private and public repositories
- To host your repository on Github, first create an account at `https://github.com/join`
- Then create an empty repository at `https://github.com/new`

Github

- Github is a provider for hosting repositories online for free.
- It allows secure access to private and public repositories
- To host your repository on Github, first create an account at `https://github.com/join`
- Then create an empty repository at `https://github.com/new`
- Don't select any of the initialization options and hit Create repository

Github

- Github is a provider for hosting repositories online for free.
- It allows secure access to private and public repositories
- To host your repository on Github, first create an account at `https://github.com/join`
- Then create an empty repository at `https://github.com/new`
- Don't select any of the initialization options and hit Create repository
- In the next tab copy the SSH link to you repository

Github

- Github is a provider for hosting repositories online for free.
- It allows secure access to private and public repositories
- To host your repository on Github, first create an account at `https://github.com/join`
- Then create an empty repository at `https://github.com/new`
- Don't select any of the initialization options and hit Create repository
- In the next tab copy the SSH link to you repository
- Other provider are available (i.e. Bitbucket, Gitlab, Stash), and the following guide will work similarly for all

SSH

- An SSH key is a cryptographic key used to secure a connection to a device

SSH

- An SSH key is a cryptographic key used to secure a connection to a device
- It is comprised of a pair of files, a public key and a private key

SSH

- An SSH key is a cryptographic key used to secure a connection to a device
- It is comprised of a pair of files, a public key and a private key
- The private key should never be shared and should always reside on your machine

SSH

- An SSH key is a cryptographic key used to secure a connection to a device
- It is comprised of a pair of files, a public key and a private key
- The private key should never be shared and should always reside on your machine
- The public key is shared with the client that would like to identify your device

SSH(cont.)

To create a SSH key pair run the following command on the bash

```
~/workspace$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/demo/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/demo/.ssh/id_rsa.
Your public key has been saved in /home/demo/.ssh/id_rsa.pub.
The key fingerprint is:
4a:dd:0a:c6:35:4e:3f:ed:27:38:8c:74:44:4d:93:67 demo@a
The key's randomart image is:
+--[ RSA 2048]----+
|          .oo.   |
|         .  o.E  |
|        + .  o   |
...
```

Make note of the path to the .pub file.

Github (cont.)

- On Github go to SSH and GPG keys settings page

Github (cont.)

- On Github go to SSH and GPG keys settings page
- Create a New SSH key

Github (cont.)

- On Github go to SSH and GPG keys settings page
- Create a New SSH key
- Copy the content of the .pub file containing your public key and Add it in the form

git remote add origin

- `git remote add origin <repository_ssh_url>` will link your local repository with a remote repo

git remote add origin

- `git remote add origin <repository_ssh_url>` will link your local repository with a remote repo
- `git push -u origin main` will syncronize the repositories by pushing the local changes, and set the default remote stream

git remote add origin

- `git remote add origin <repository_ssh_url>` will link your local repository with a remote repo
- `git push -u origin main` will syncronize the repositories by pushing the local changes, and set the default remote stream
- You will be promted to verify the fingerprint and accept the connection

git remote add origin

- `git remote add origin <repository_ssh_url>` will link your local repository with a remote repo
- `git push -u origin main` will syncronize the repositories by pushing the local changes, and set the default remote stream
- You will be promted to verify the fingerprint and accept the connection
- If you prefer using HTTPS and 2FA on github, instead of SSH keys visit the official documentation

git push

- `git push` is the command used to update the remote repository

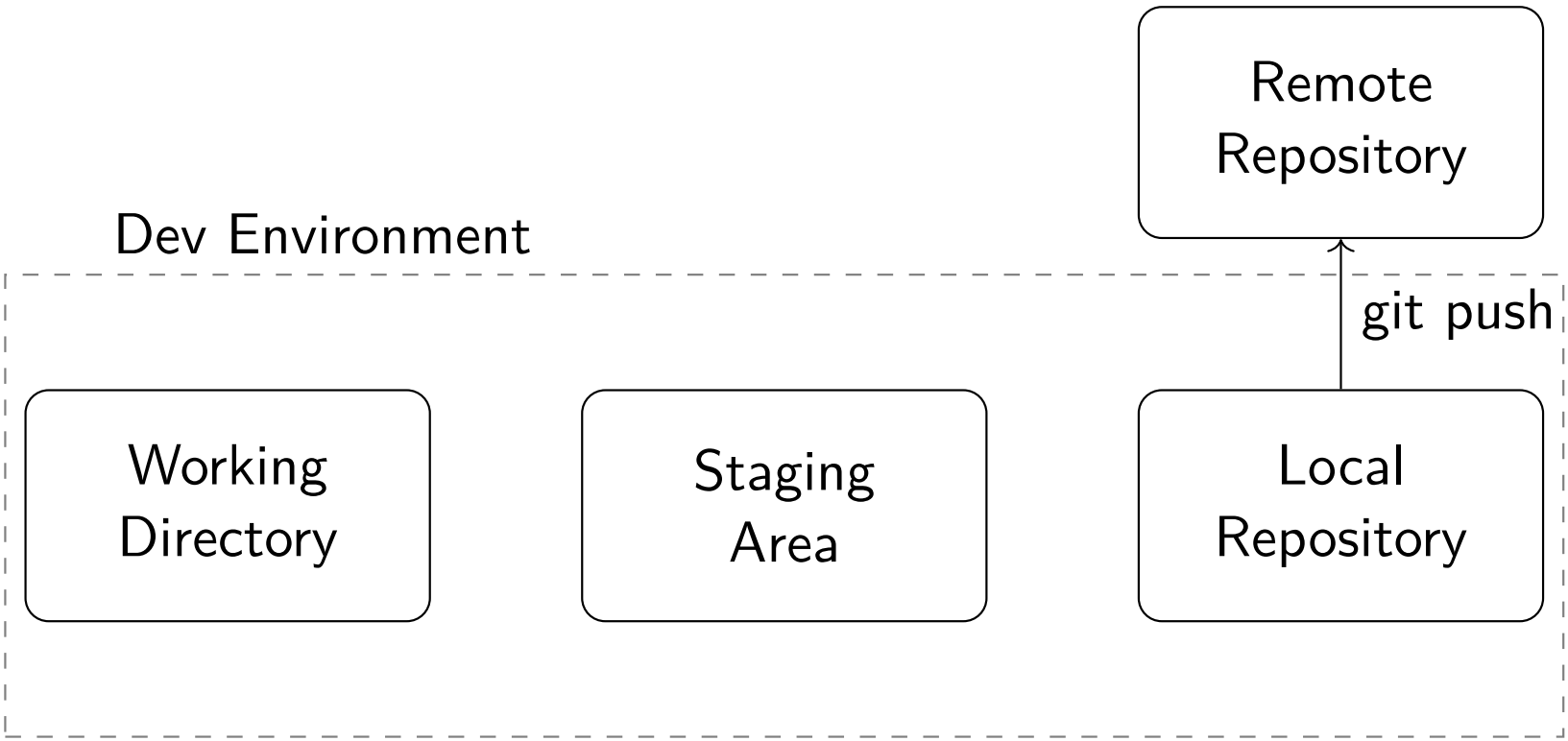
git push

- `git push` is the command used to update the remote repository
- When there are one or several commit on the local repository ready to be shared, they have to be pushed to the remote repository

git push

- `git push` is the command used to update the remote repository
- When there are one or several commit on the local repository ready to be shared, they have to be pushed to the remote repository
- `git push <remote> <branch>` allows to push the content of a local branch to a matching remote branch (i.e `git push origin main`)

git push (cont.)



git fetch

- `git push` fails if the local repository is behind the remote one

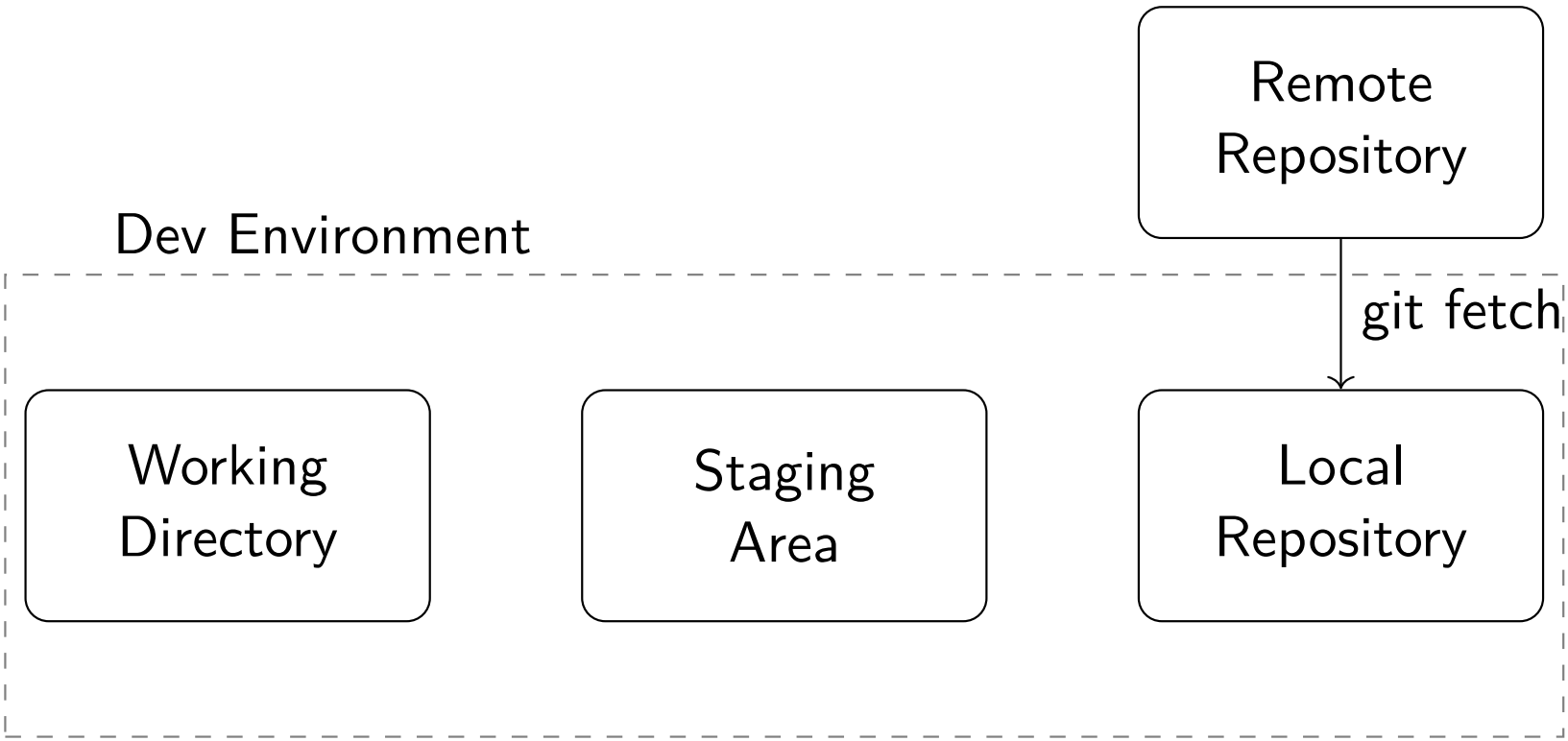
git fetch

- `git push` fails if the local repository is behind the remote one
- `git fetch` allows to pull commits, files and refs from the remote repository without merging with the local

git fetch

- `git push` fails if the local repository is behind the remote one
- `git fetch` allows to pull commits, files and refs from the remote repository without merging with the local
- `git diff <branch> origin/<branch>` will show the difference between a local branch and a remote one, after fetching the metadata

git fetch (cont.)



git pull

- `git pull` will fetch the files and also merge them with the current state of your workspace

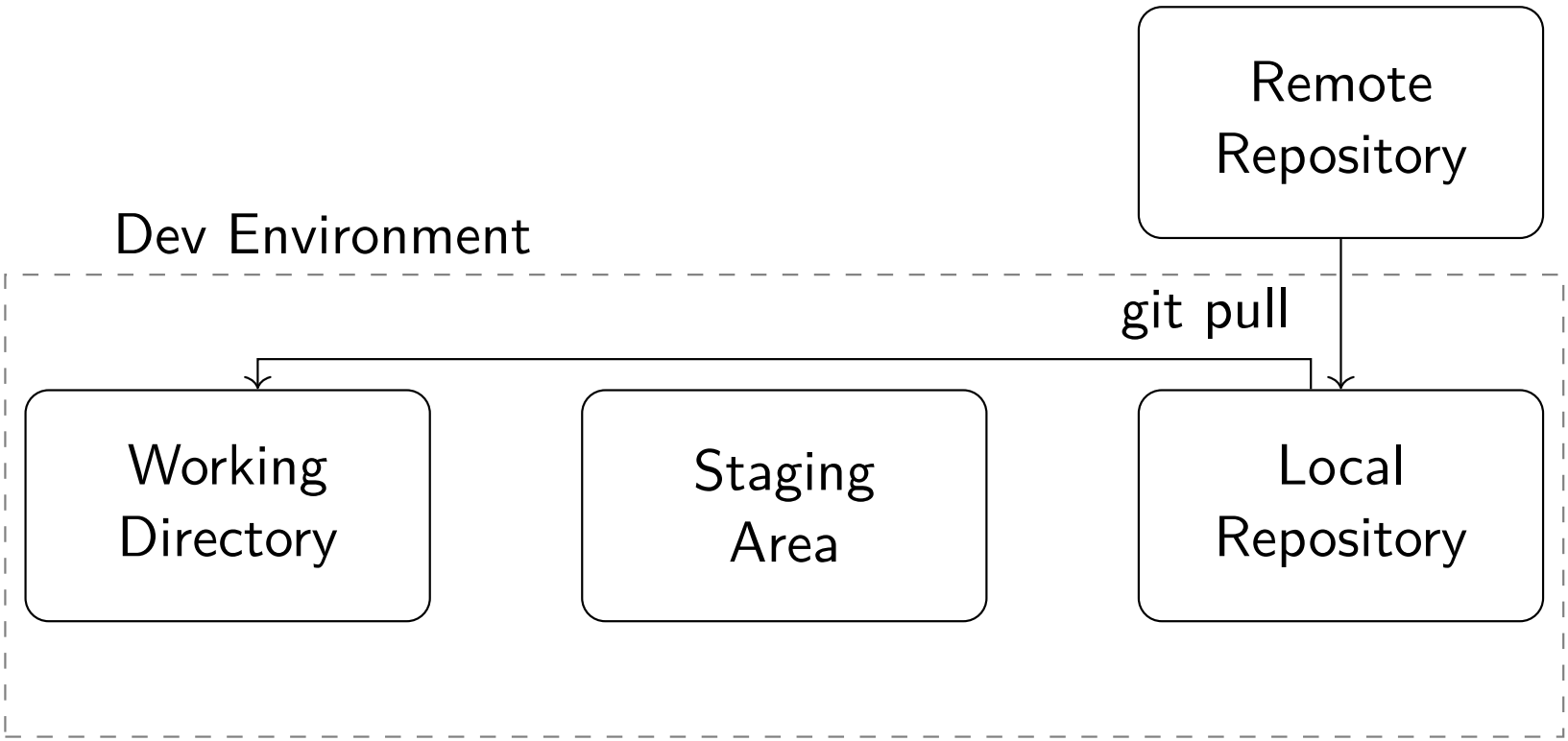
git pull

- `git pull` will fetch the files and also merge them with the current state of your workspace
- It can cause Merge conflicts, that need to be fixed before pushing back to the remote

git pull

- `git pull` will fetch the files and also merge them with the current state of your workspace
- It can cause Merge conflicts, that need to be fixed before pushing back to the remote
- The complete remote workflow will consist of one or more commits, fetching, pulling and merging conflicts, and finally pushing to the remote

git pull (cont.)



git clone

- A remote repository can be shared between multiple users

git clone

- A remote repository can be shared between multiple users
- To download locally the content of a remote repository you can use `git clone <remote_url>`

git clone

- A remote repository can be shared between multiple users
- To download locally the content of a remote repository you can use `git clone <remote_url>`
- Cloning and pulling is usually permitted on public repositories, but pushing changes is restricted to users added as collaborators on Github

git clone

- A remote repository can be shared between multiple users
- To download locally the content of a remote repository you can use `git clone <remote_url>`
- Cloning and pulling is usually permitted on public repositories, but pushing changes is restricted to users added as collaborators on Github
- From your repository web page go to Settings > Manage access > Invite a collaborator