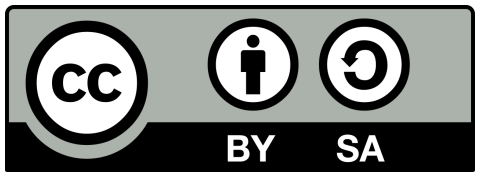
Git for beginners

Module 2

Gianfranco Gallizia

2025



Rebase.

Recap

Git creates snapshots of the tracked files and calculates the SHA-1 hash of these snapshots These hashes identify the right binary file (*blob*) inside the `GIT_DIR`.

Git creates snapshots of the tracked files and calculates the SHA-1 hash of these snapshots These hashes identify the right binary file (*blob*) inside the `GIT_DIR`.

A commit is an entry containing the metadata linked to a specific change:

Rebase.

Recap

Git creates snapshots of the tracked files and calculates the SHA-1 hash of these snapshots These hashes identify the right binary file (*blob*) inside the `GIT_DIR`.

A commit is an entry containing the metadata linked to a specific change:

- Author's name.
- Author's email.
- Date and time of commit's creation.
- SHA-1 of the blob.
- SHA-1 of the parent commit(s).
- SHA-1 of all the previous metadata.

Rebase.

Recap

Tree

A system of chained commits from which it is possible to go back to the first commit entered, regardless of the starting point.

Rebase.

Recap

Tree

A system of chained commits from which it is possible to go back to the first commit entered, regardless of the starting point.

Nota bene:

The root of the tree is **the last commit inserted** (`HEAD`), not the first.

Rebase.

Recap

Tree

A system of chained commits from which it is possible to go back to the first commit entered, regardless of the starting point.

Nota bene:

The root of the tree is **the last commit inserted** (`HEAD`), not the first.

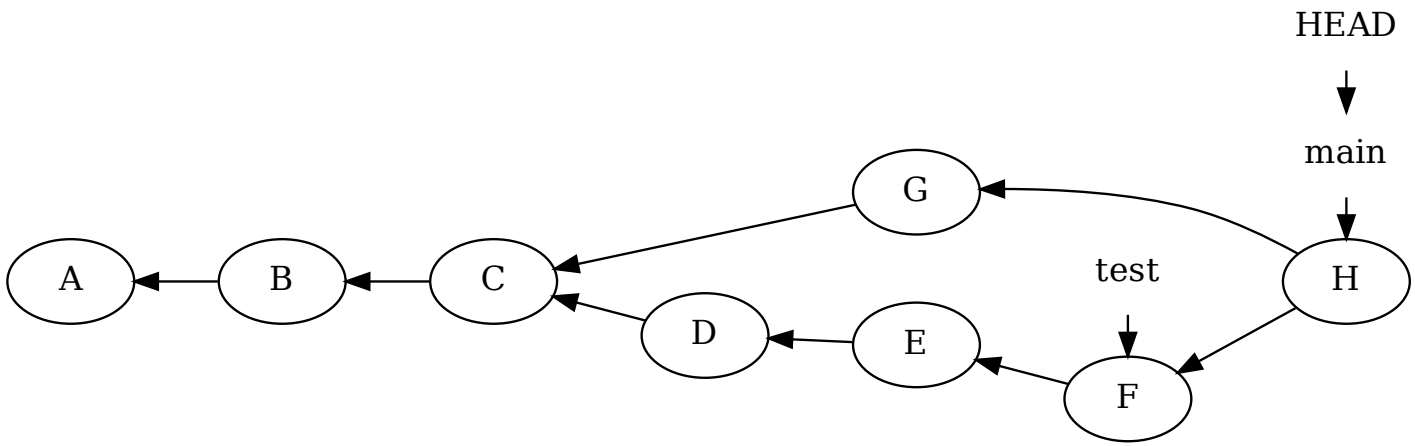
A commit creates a new root in the tree, whereas a merge combines two separate sub-trees by giving them a new shared root.

Git's standard approach is to have developers work independently on their own sub-trees, then combine their work through merges. During a merge, any conflicting changes are reconciled and resolved in the resulting commit.

Rebase.

Recap

Git's standard approach is to have developers work independently on their own sub-trees, then combine their work through merges. During a merge, any conflicting changes are reconciled and resolved in the resulting commit.



Rebase.

Definition

Rebase.

Definition

Rebase

Reposition the development branch so that it begins from the latest point in the main branch.

Rebase.

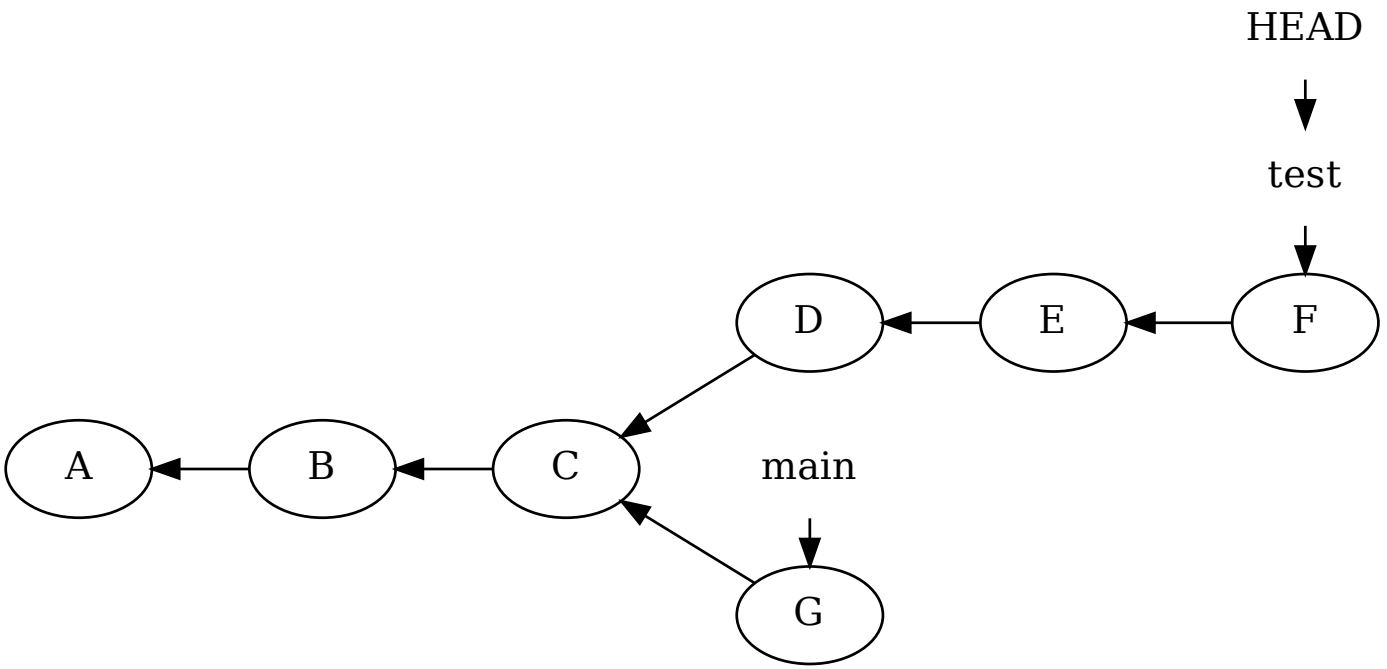
Definition

Let's go back:

Rebase.

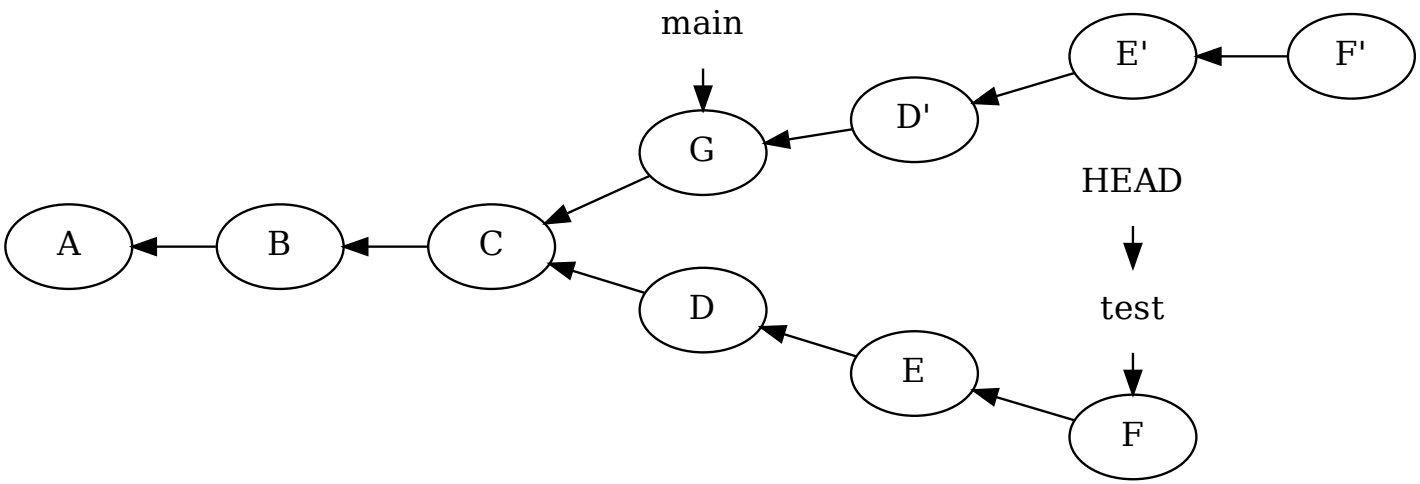
Definition

Let's go back:



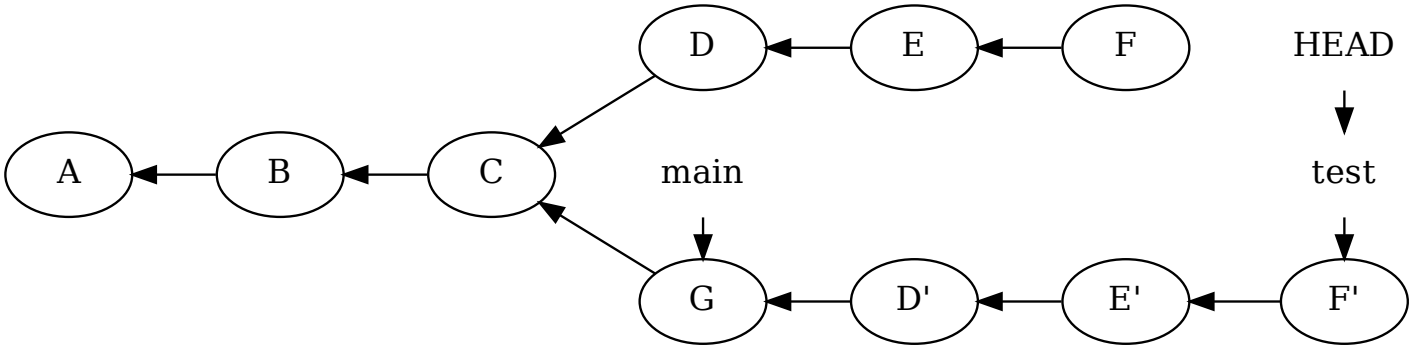
Rebase.

Definition



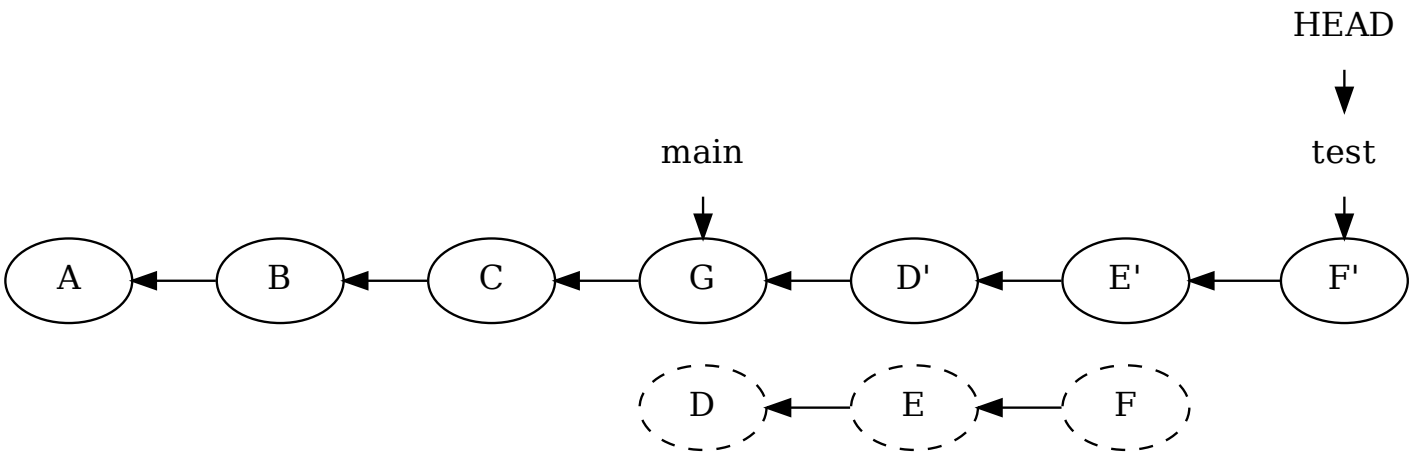
Rebase.

Definition



Rebase.

Definition



Rebase.

Issues

Conflicts

In case of conflicting changes git will stop the rebase:

```
git rebase main
Auto-merging README.txt
CONFLICT (content): Merge conflict in README.txt
error: could not apply cf21f66... Lowercase.
[omissis...]
Could not apply cf21f66... Lowercase.
```

Conflicts

Git also provides some suggestions on how to proceed:

```
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".

To abort and get back to the state before "git rebase", run "git rebase --abort".
```

Rebase.

Issues

WARNING

Rebasing **rewrites the history of a branch**. This isn't a problem for a local branch that only exists on our machine, but it becomes a major issue once we've already distributed our changes to others.

To perform a rebase:

To perform a rebase:

1. Switch to the branch **that you want to rebase** using

```
git checkout $BRANCH
```

To perform a rebase:

1. Switch to the branch **that you want to rebase** using

   ```
   git checkout $BRANCH
   ```

2. Run the command `git rebase $BASE`, where `$BASE` is the branch that you want to use as the new starting point.

Rebase.

How to perform a rebase

To perform a rebase:

1. Switch to the branch **that you want to rebase** using

   ```
   git checkout $BRANCH
   ```

2. Run the command `git rebase $BASE`, where `$BASE` is the branch that you want to use as the new starting point.
3. Wait for the rebase to complete or for it to be interrupted due to conflicts.

Rebase.

How to perform a rebase

To perform a rebase:

1. Switch to the branch **that you want to rebase** using

   ```
   git checkout $BRANCH
   ```

2. Run the command `git rebase $BASE`, where `$BASE` is the branch that you want to use as the new starting point.
3. Wait for the rebase to complete or for it to be interrupted due to conflicts.
4. If the rebase is interrupted, **resolve the conflicts** and then continue the rebase with

   ```
   git rebase --continue
   ```

Sharing your work.

Patches

My collaborators don't use git...

My collaborators don't use git...

```
PATCH(1)      General Commands Manual      PATCH(1)

NAME
       patch - apply a diff file to an original

SYNOPSIS
       patch [options] [originalfile [patchfile]]
```

Sharing your work.

Patches

Sending and receiving patches was the historical method of collaboration between developers.

Sharing your work.

Patches

Sending and receiving patches was the historical method of collaboration between developers.

The main advantage of this method is that you are not tied to a specific Version Control System. The resulting files can be read, sent to the `patch` command and integrated without the need for the receiver to install git.

Sharing your work.

Patches

Sending and receiving patches was the historical method of collaboration between developers.

The main advantage of this method is that you are not tied to a specific Version Control System. The resulting files can be read, sent to the `patch` command and integrated without the need for the receiver to install git.

The main disadvantage is that it's an obsolete way of collaboration without authentication and it's based on mutual trust between people. Furthermore those receiving the patches must do some work to integrate those changes to their codebase.

Sharing your work.

Git through SSH

SSH

Network protocol at the Application level of the TCP/IP stack usable by git to send data between hosts.

SSH

Network protocol at the Application level of the TCP/IP stack usable by git to send data between hosts.
Big emphasis on security: through SSH all the communications are authenticated (both the user and the machines are checked) and performed via encrypted tunnels over the network according to a set of algorithms negotiated between the hosts.

Sharing your work.

Git through SSH

The basic idea

Create a repository on an host with SSH active, instruct the clients to send their changes through SSH to said host and to retrieve our changes though SSH from that host.

Sharing your work.

Git through SSH

The basic idea

Create a repository on an host with SSH active, instruct the clients to send their changes through SSH to said host and to retrieve our changes though SSH from that host.

Nota bene:

The host can be any machine and theoretically all the developers could open the access to their machines through SSH and send data between them in a peer-to-peer network.

Server configuration

Server configuration

We will not do it, if you are interested there's a whole chapter dedicated to the server setup for git in the Git Book (https://git-scm.com/book/en/v2/)

Client configuration

Client configuration

If you want to collaborate to the development of a project that already uses git through SSH follow these steps:

Client configuration

If you want to collaborate to the development of a project that already uses git through SSH follow these steps:

1. Ask for an account; or register with the organization/git service provider and then ask for access.

Client configuration

If you want to collaborate to the development of a project that already uses git through SSH follow these steps:

1. Ask for an account; or register with the organization/git service provider and then ask for access.
2. Obtain the repository URL in the format

   `ssh://user@git.example.com/org/project.git`

Client configuration

If you want to collaborate to the development of a project that already uses git through SSH follow these steps:

1. Ask for an account; or register with the organization/git service provider and then ask for access.
2. Obtain the repository URL in the format

   `ssh://user@git.example.com/org/project.git`
3. Use the `git clone $URL` command (where `$URL` is the one previously obtained).

Client configuration (cont.)

`git clone` will connect through SSH to `git.example.com` and then start downloading the repository in a directory called `project`. Once finished you will have a local copy of the whole repository and your working copy will be automatically placed on the last commit of the default branch.

Sharing your work.

Git through SSH - git clone

Cloning in a specific directory:

```
git clone $URL $WORKDIR
```

Sharing your work.

Git through SSH - git clone

Cloning in a specific directory:

```
git clone $URL $WORKDIR
```

Cloning just the last commit:

```
git clone --depth 1 $URL
```

Sharing your work.

Git through SSH - git clone

Cloning in a specific directory:

```
git clone $URL $WORKDIR
```

Cloning just the last commit:

```
git clone --depth 1 $URL
```

Cloning just the last commit of a specific branch:

```
git clone --single-branch -b $BRANCH --depth 1 $URL
```

Cloning a remote repository keeping separate `GIT_DIR` and `WORKDIR`:

```
git clone --separate-git-dir=$GIT_DIR $URL $WORKDIR
```

Sharing your work.

Git through SSH - git clone

Cloning a remote repository keeping separate `GIT_DIR` and `WORKDIR`:

```
git clone --separate-git-dir=$GIT_DIR $URL $WORKDIR
```

Cloning only `GIT_DIR` get a *bare repository*:

```
git clone --bare $URL $GIT_DIR
```

Sharing your work.

Git through SSH - from local to remote

Remote

In git a *remote* is an URL pointing to a remote repository. Git can have more than one remote per local repository.
For each local repository there can only be one *upstream* remote. The *upstream* remote is the default remote for the data transfer operations.

Connecting to a remote

Connecting to a remote

1. Add a remote.

Sharing your work.

Git through SSH - from local to remote

Connecting to a remote

1. Add a remote.
2. Send your changes.

Sharing your work.

Git through SSH - from local to remote

Connecting to a remote

1. Add a remote.
2. Send your changes.
3. (Optional) send your tags.

Sharing your work.

Git through SSH - from local to remote

Connecting to a remote

1. Add a remote.
2. Send your changes.
3. (Optional) send your tags.

```
git remote add $REMOTE_NAME $URL
git push --all --set-upstream $REMOTE_NAME
git push --tags
```

Tags

You can assign a human readable pointer to a commit (*tag a commit*). Usually this operation is performed by the maintainer to indicate which commit has been used to create a specific version of the software.

Tags

You can assign a human readable pointer to a commit (*tag a commit*). Usually this operation is performed by the maintainer to indicate which commit has been used to create a specific version of the software.

Assigning a tag:

```
git tag $LABEL $COMMIT
```

Sharing your work.

Interlude - specify a commit

Ways to specify a commit:

Ways to specify a commit:

Full hash: `284a5a91c43d7a2e0ceb1eb4721dca08da17b3a0`

Ways to specify a commit:

Full hash: `284a5a91c43d7a2e0ceb1eb4721dca08da17b3a0`

Shortened hash: `284a5a91`

Ways to specify a commit:

Full hash: `284a5a91c43d7a2e0ceb1eb4721dca08da17b3a0`
Shortened hash: `284a5a91`
Last commit: `HEAD`

Ways to specify a commit:

Full hash: `284a5a91c43d7a2e0ceb1eb4721dca08da17b3a0`
Shortened hash: `284a5a91`
Last commit: `HEAD`
Branch name: `main`

Ways to specify a commit:

Full hash: `284a5a91c43d7a2e0ceb1eb4721dca08da17b3a0`
Shortened hash: `284a5a91`
Last commit: `HEAD`
Branch name: `main`
Relative (3 commits before main): `main^3` or `main~3`

Sharing your work.

Retrieve other people changes

Retrieve other people changes

First we make sure we are ready to retrieve the changes from the remote repository. To do so we will use the `git status` command:

Retrieve other people changes

First we make sure we are ready to retrieve the changes from the remote repository. To do so we will use the `git status` command:

```
git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Sharing your work.

Retrieve other people changes

Sharing your work.

Retrieve other people changes

Retrieve from upstream remote (default):

```
git pull
```

Sharing your work.

Retrieve other people changes

Retrieve from upstream remote (default):

```
git pull
```

Retrieve from a different remote:

```
git pull $REMOTE_NAME $REMOTE_BRANCH
```

Sharing your work.

Interlude - pull strategy

Pull strategy

Unless otherwise specified; `git pull` will try to *merge* the remote branch into the local branch.

Pull strategy

Unless otherwise specified; `git pull` will try to *merge* the remote branch into the local branch.
When possible git will perform a *fast forward*, otherwise it will create a *merge commit*.

Pull strategy

Unless otherwise specified; `git pull` will try to *merge* the remote branch into the local branch.
When possible git will perform a *fast forward*, otherwise it will create a *merge commit*.
If you want to apply your local changes on top of the remote changes you can opt for the *rebase* strategy:

```
git pull --rebase
```

Pull strategy

Unless otherwise specified; `git pull` will try to *merge* the remote branch into the local branch.
When possible git will perform a *fast forward*, otherwise it will create a *merge commit*.
If you want to apply your local changes on top of the remote changes you can opt for the *rebase* strategy:

```
git pull --rebase
```

Nota bene:

Everything we said regarding merge, conflicts and rebase apply here.

Sharing your work.

GitHub Workflow

Collaborations

Each project has its own rules and it's hard to give some recommendations that are generic enough to be applicable everywhere. Except for the very basic ones.

Sharing your work.

GitHub Workflow

Collaborations

Each project has its own rules and it's hard to give some recommendations that are generic enough to be applicable everywhere. Except for the very basic ones.

GitHub

GitHub is the largest git service provider in the World.
Each user registered on GitHub could create a repository on GitHub's servers and access through SSH or HTTPS.

GitHub Workflow

GitHub Workflow

1. Find the project's page on GitHub.

GitHub Workflow

1. Find the project's page on GitHub.
2. Press the "fork" button to create a copy of the project's repository under your account.

GitHub Workflow

1. Find the project's page on GitHub.
2. Press the "fork" button to create a copy of the project's repository under your account.
3. Clone locally your *fork*.

GitHub Workflow

1. Find the project's page on GitHub.
2. Press the "fork" button to create a copy of the project's repository under your account.
3. Clone locally your *fork*.
4. Make your changes, create your branches and work to the project just like it was yours.

GitHub Workflow

1. Find the project's page on GitHub.
2. Press the "fork" button to create a copy of the project's repository under your account.
3. Clone locally your *fork*.
4. Make your changes, create your branches and work to the project just like it was yours.
5. When you're satisfied with your work push it on GitHub.

GitHub Workflow

1. Find the project's page on GitHub.
2. Press the "fork" button to create a copy of the project's repository under your account.
3. Clone locally your *fork*.
4. Make your changes, create your branches and work to the project just like it was yours.
5. When you're satisfied with your work push it on GitHub.
6. Open a "Pull Request" to the project's maintainers and ask them to integrate your changes to the main project.

Sharing your work.

GitHub Workflow

Pro

GitHub allows to create an unlimited number of public and private repositories provided that you respect the Terms and Conditions set by Microsoft.

Sharing your work.

GitHub Workflow

Pro

GitHub allows to create an unlimited number of public and private repositories provided that you respect the Terms and Conditions set by Microsoft.

Con

Almost all the data flows through GitHub servers: all the communications pass by GitHub web interface and all the code is shared via GitHub (regardless if it's integrated to the main project or not).

Help!

I made a mess!

Help! I made a mess!

Help! I made a mess!

1. Don't panic.

Help! I made a mess!

1. Don't panic.
2. If you have not issued `git push` you can still fix it (usually).

Help! I made a mess!

1. Don't panic.
2. If you have not issued `git push` you can still fix it (usually).
3. Everything that is put in `.gitignore` is under our responsibility (if we delete it: it's lost).

Help!

I made a mess!

Help! I made a mess!

1. Don't panic.
2. If you have not issued `git push` you can still fix it (usually).
3. Everything that is put in `.gitignore` is under our responsibility (if we delete it: it's lost).
4. Everything that is not tracked by git is under our responsibility (if we delete it: it's lost).

Help!

Amend a commit

I wrote the wrong commit message!

Help!

Amend a commit

I wrote the wrong commit message!

```
git commit --amend
```

Help!

Amend a commit

I've put the wrong content inside the commit!

Help!

Amend a commit

I've put the wrong content inside the commit!

```
git reset --soft HEAD^1
# Edit the file with a text editor
git add -i
git commit
```

I've committed to the wrong branch!

Help!

Amend a commit

I've committed to the wrong branch!

```
git log --oneline
git checkout dev
git cherry-pick $HASH_OF_THE_LAST_MAIN_COMMIT
git checkout main
git reset --hard HEAD^1
```

I've committed to the wrong branch!

```
git log --oneline
git checkout dev
git cherry-pick $HASH_OF_THE_LAST_MAIN_COMMIT
git checkout main
git reset --hard HEAD^1
```

IMPORTANT!

`git reset --soft HEAD^1` deletes the last commit of the current branch and leaves the working copy unaltered. `git reset --hard HEAD^1` deletes the last commit of the current branch **AND** the relative changes from working copy.

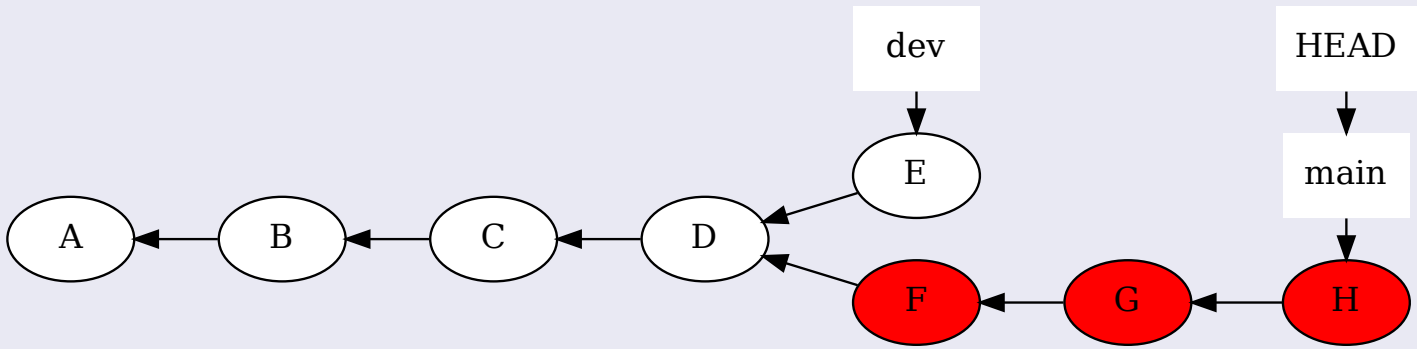
Help!

A complex example

Help!

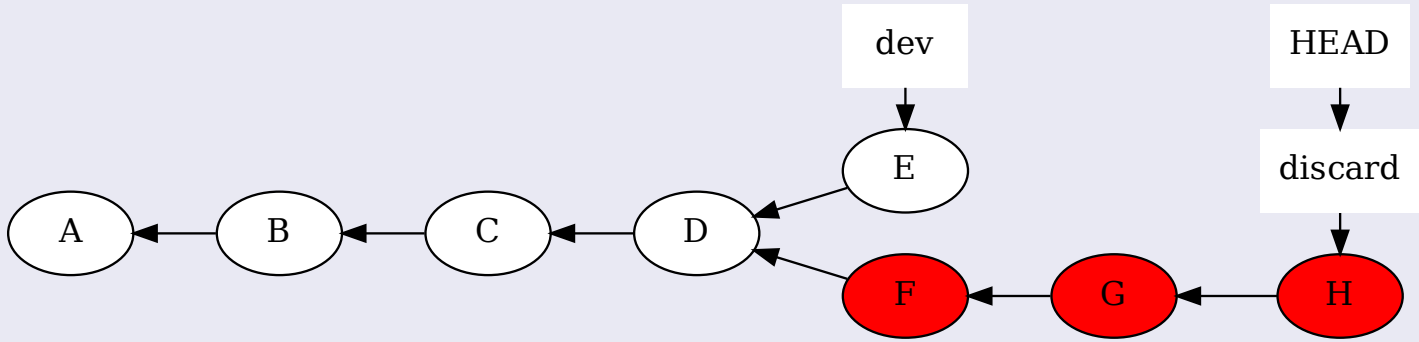
A complex example

I made a bunch of commits on the wrong branch!



Rename `main` in `discard`

```
git branch -m discard
```

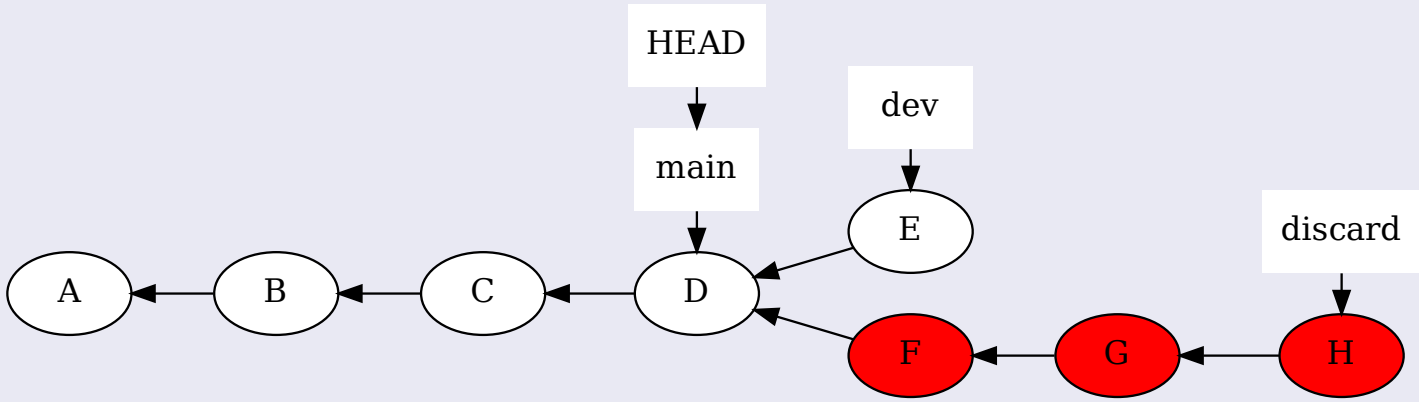


Help!

A complex example

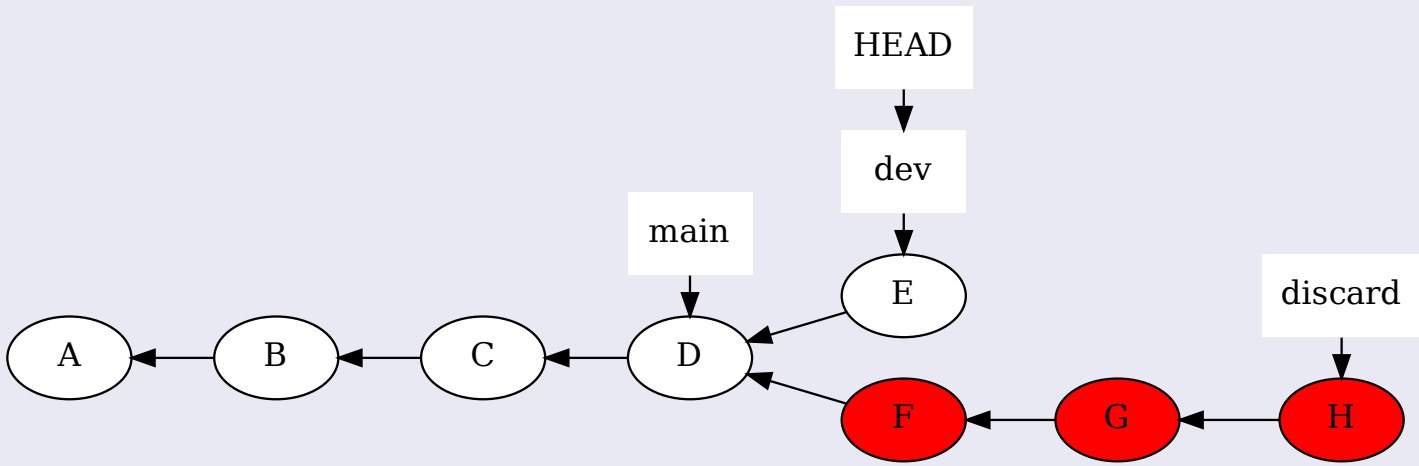
Create a new branch called `main` ending in `D`

```
git checkout $D
git branch main
```



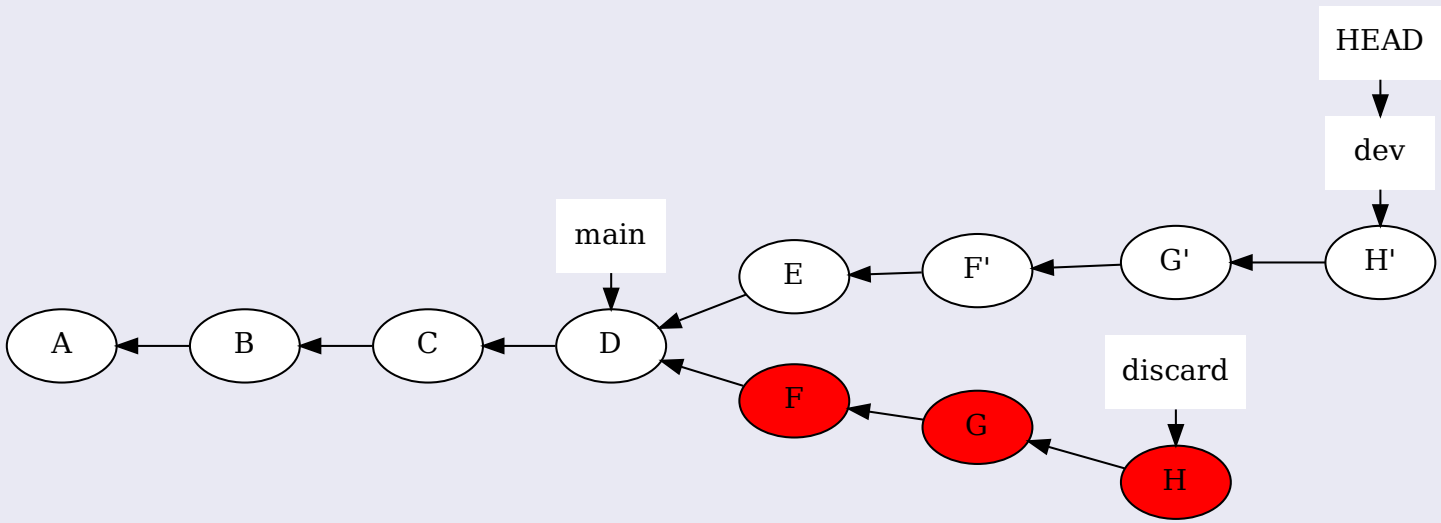
Let's move on dev

```
git checkout dev
```



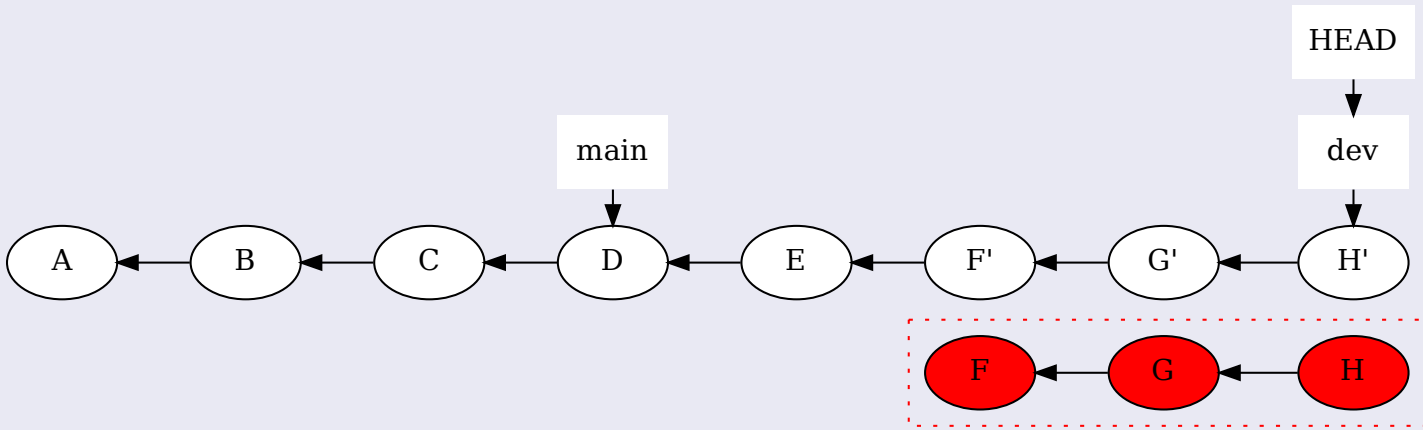
Let's copy `F`, `G` and `H` in `dev`

```
git cherry-pick main..discard
```



Delete the `discard` branch

```
git branch -D discard
```



Help!

Pull rebase

Help!

Pull rebase

I've committed on the default branch before pulling the remote changes!

Help!

Pull rebase

I've committed on the default branch before pulling the remote changes!

```
git pull --rebase
```

Help!

After a push

I pushed something I shouldn't have pushed...

Help!

After a push

I pushed something I shouldn't have pushed...

I'm sorry. Create a new commit where you fix everything, write "Revert wrong modifications." as the commit message, issue `git push` and embrace the shame.

Best practices.

Things to do

Best practices.

Things to do

git status mantra

Best practices.

Things to do

git status mantra

Force yourself to issue `git status` **before and after** issuing another git command.

Best practices.

Things to do

git status mantra

Force yourself to issue `git status` **before and after** issuing another git command.
`git status` shows a snapshot of your working copy, which files you have placed in the staging area to be committed, and also suggests what to do during merge and rebase operations.

Best practices.

Things to do

Local branches are free

Local branches are free

If you're unsure about issuing `git checkout -b new_branch` before moving on: create the branch.

Best practices.

Things to do

Local branches are free

If you're unsure about issuing `git checkout -b new_branch` before moving on: create the branch.

Branching in other VCSs is an epic event reserved to a few blessed individuals, branching in git is something that's done multiple times per day sometimes.

Best practices.

Things to do

`pull` comes always before `push`

`pull` comes always before `push`

1. Check out a local branch.

`pull` comes always before `push`

1. Check out a local branch.
2. Make your changes.

Best practices.

Things to do

`pull` comes always before `push`

1. Check out a local branch.
2. Make your changes.
3. Go back to the starting branch.

Best practices.

Things to do

`pull` comes always before `push`

1. Check out a local branch.
2. Make your changes.
3. Go back to the starting branch.
4. `git pull`

Best practices.

Things to do

`pull` comes always before `push`

1. Check out a local branch.
2. Make your changes.
3. Go back to the starting branch.
4. `git pull`
5. `git log --all --decorate --oneline --graph`

Best practices.

Things to do

`pull` comes always before `push`

1. Check out a local branch.
2. Make your changes.
3. Go back to the starting branch.
4. `git pull`
5. `git log --all --decorate --oneline --graph`
6. Decide if you want to merge the local branch, rebase the local branch or `git cherry-pick`, etc. etc.

Bonus: be careful with aliases

Best practices.

Things to do

Bonus: be careful with aliases

```
git config --global alias.adog "log --all \
--decorate --oneline --graph"
git adog
*   8b393c8 (HEAD -> change_B) Merge branch\ldots
|\
| * 5855fe7 (change_A) Lowercase.
* | bd1130e (main) Double the letters.
|/
* d1f0f78 Initial Commit.
```

Best practices.

Things to avoid

Best practices.

Things to avoid


```
git commit -a -m '.'
```

Best practices.

Things to avoid


```
git commit -a -m '.'
```

"I don't care, just save it."

Best practices.

Things to avoid

`git push --force`

Best practices.

Things to avoid

`git push --force`

Git refuses to complete the push action if it detects some unreconcilable differences between the local repository and the remote repository.

Best practices.

Things to avoid

`git push --force`

Git refuses to complete the push action if it detects some unreconcilable differences between the local repository and the remote repository.

`git push --force` tels git to continue with the operation and to **overwrite** the changes with those present in the local branch. Basically you're being a dictator and you are implicitly telling your collaborators that their work is not as important as yours.

Best practices.

Use case - LADE

How Git is used in LADE

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.
- When their changes are ready they open a **merge request** on Gitlab.

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.
- When their changes are ready they open a **merge request** on Gitlab.
- The person responsible evaluates the merge request and can do one of the following actions:

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.
- When their changes are ready they open a **merge request** on Gitlab.
- The person responsible evaluates the merge request and can do one of the following actions:
  - Accept the merge request as is.

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.
- When their changes are ready they open a **merge request** on Gitlab.
- The person responsible evaluates the merge request and can do one of the following actions:
  - Accept the merge request as is.
  - Give some suggestions in the form of comments to the merge request.

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.
- When their changes are ready they open a **merge request** on Gitlab.
- The person responsible evaluates the merge request and can do one of the following actions:
  - Accept the merge request as is.
  - Give some suggestions in the form of comments to the merge request.
  - Modify the merge request by pushing their changes to the merge request branch.

Best practices.

Use case - LADE

How Git is used in LADE

- Each repository has a person responsible for that repository.
- Each member of LADE can create additional branches.
- When their changes are ready they open a **merge request** on Gitlab.
- The person responsible evaluates the merge request and can do one of the following actions:
  - Accept the merge request as is.
  - Give some suggestions in the form of comments to the merge request.
  - Modify the merge request by pushing their changes to the merge request branch.
  - Refuse the merge request and close it.

Best practices.

Use case - LADE

Pro

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.
- There's a check on what gets into the main branch.

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.
- There's a check on what gets into the main branch.
- There are no doubts on *what* is the repository to clone.

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.
- There's a check on what gets into the main branch.
- There are no doubts on *what* is the repository to clone.

Con

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.
- There's a check on what gets into the main branch.
- There are no doubts on *what* is the repository to clone.

Con

- The person responsible for the repository must resolve the conflicts or convince the proposer to do it.

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.
- There's a check on what gets into the main branch.
- There are no doubts on *what* is the repository to clone.

Con

- The person responsible for the repository must resolve the conflicts or convince the proposer to do it.
- History is not linear.

Best practices.

Use case - LADE

Pro

- People don't get stuck waiting for their changes to be approved.
- There's a check on what gets into the main branch.
- There are no doubts on *what* is the repository to clone.

Con

- The person responsible for the repository must resolve the conflicts or convince the proposer to do it.
- History is not linear.
- If I have an open merge request and my changes are not part of the main branch I must merge from the main branch, I cannot rebase.

End of module 2

Thank you for your attention!

End of module 2

Thank you for your attention!

Any questions?