git - Part II
Branching & Remotes

Stefan Pranger

stefan.pranger@student.tugraz.at

03. November 2020

Overview

1. Remote Repositories
2. Branches
   - Working with branches
   - Merging and merge conflicts
3. Stash
4. Reflog
5. Remotes
6. Summary

Road Map

- 02. 11.: Local Repository and git basics
- *Today*: Branches, Remotes, non-linear workflow and working in a team
- 04. 11.: Miscellaneous: Tagging, Reflog, .gitconfig, Best Practices and Workflows, Hooks, Background, etc. . .

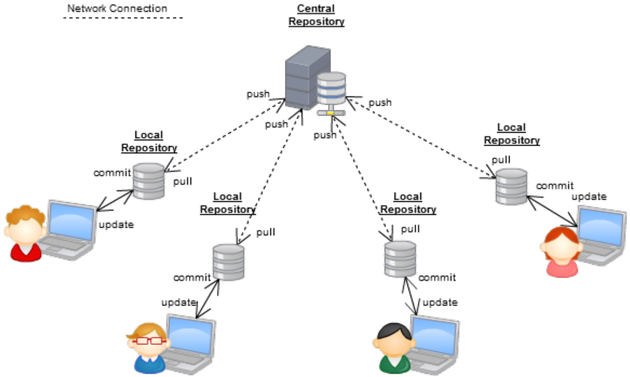
Remote Repositories

Remote

```
$> git pull origin main
```

Branch

Remote Repository



```
$> git pull origin main
```

Recap

- Local Repository
- Stages
- Commits

  i.e. one entry in the history of your project

Untracked Modified Staged Committed

Recap

- Local Repository
- Stages
- Commits

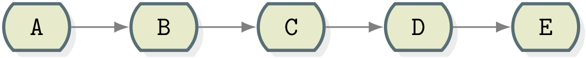
  i.e. one entry in the *history* of your project

Quoting Wikipedia:

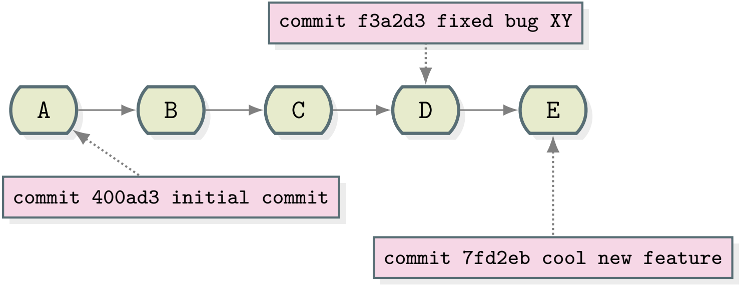
> … support for *distributed*, *non-linear workflows*.

`Untracked` `Modified` `Staged` `Committed`

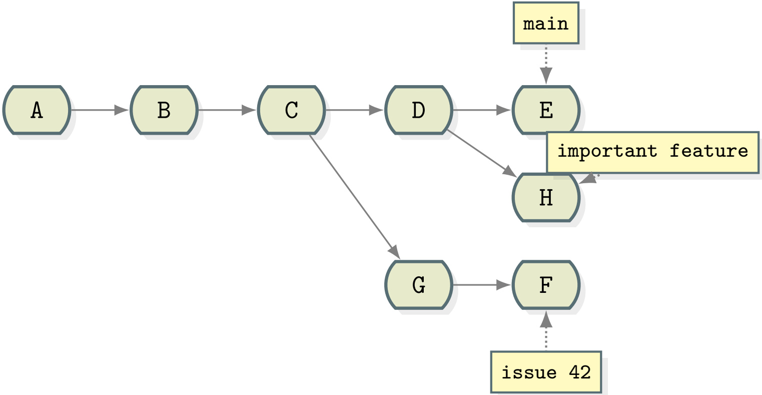
What is a branch?



What is a branch?


commit f3a2d3 fixed bug XY
A
B
C
D
E
commit 400ad3 initial commit
commit 7fd2eb cool new feature

What is a branch?



What is a branch?

- A new working directory
- with the same version of files
- and its own set of commits

What is a branch?

- A new working directory
- with the same version of files
- and its own set of commits

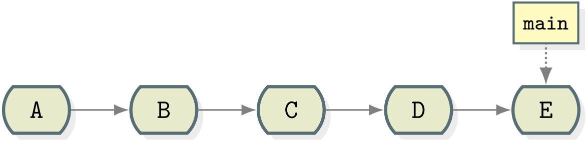
Allows you to apply changes only within the new working directory!

What is a branch?

- A new working directory
- with the same version of files
- and its own set of commits

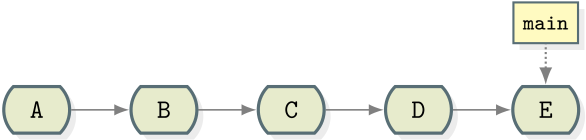
Allows you to apply changes only within the new working directory!

- internally a reference to a particular commit



Branching

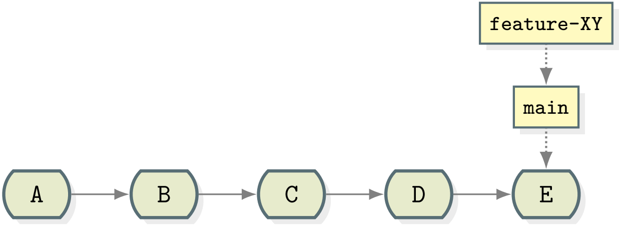
Creating a new branch:



Branching

Creating a new branch:

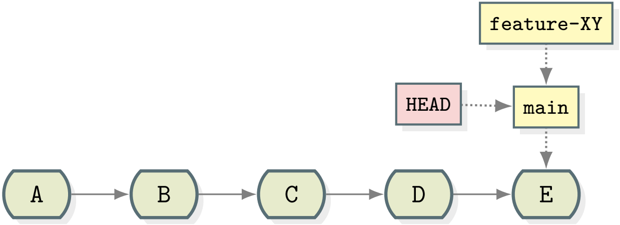
- `$> git branch <branchname>`



Branching

Creating a new branch:

- `$> git branch <branchname>`



Branching

Creating a new branch:

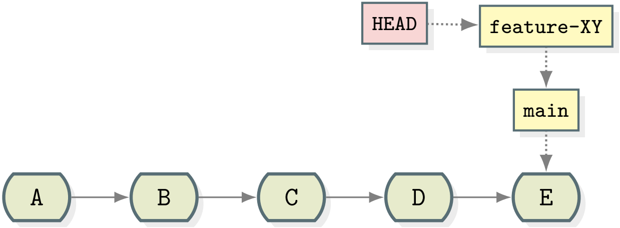
- `$> git branch <branchname>`

Switching branches:

- `$> git switch feature-XY` , `$> git switch main`

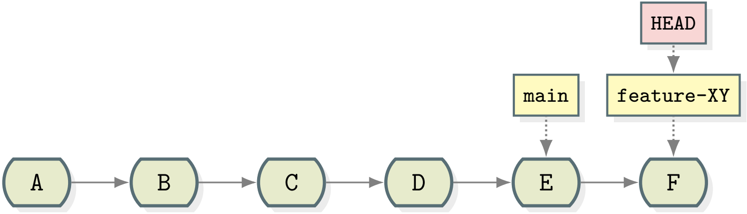
Creating and switching:

- `$> git switch -c feature-XY`



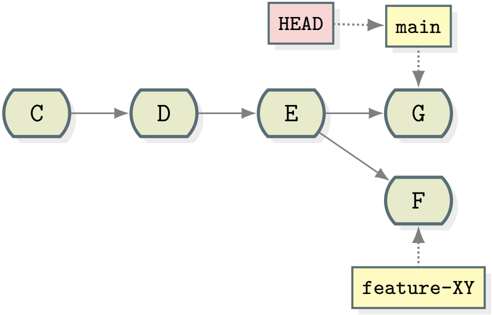
Branching

After one `$> git commit -m "<message>"`:



Branching

We can also switch back to `main` and commit there:



Branching

- Comparing the logs

  Two ”different” histories!

Branching

- Comparing the logs

  Two "different" histories!

- We can compare these histories with

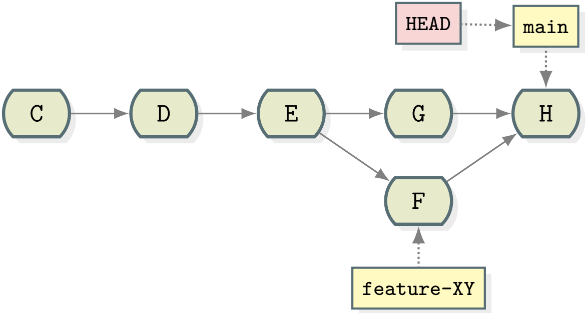
```
$> git diff <branch1> <branch2>
```

Branching

- Comparing the logs
  Two "different" histories!
- We can compare these histories with

```
$> git diff <branch1> <branch2>
```

- `main` needs to integrate the information from the feature branch!



What does a merge do?

- Integrates commits from one branch into another

What does a merge do?

- Integrates commits from one branch into another
- `$> git merge <branchname>`

What does a merge do?

- Integrates commits from one branch into another
-
```
$> git merge <branchname>
```
- Final step when implementing a new feature.

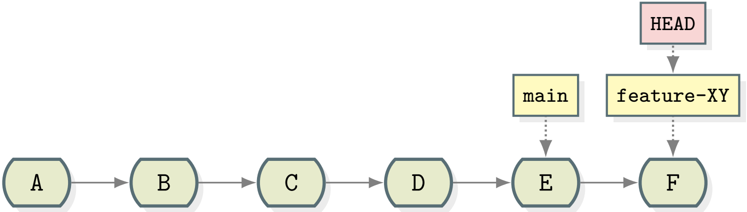
What does a merge do?

- Integrates commits from one branch into another
- `$> git merge <branchname>`
- Final step when implementing a new feature.
- Biggest source for problems *(merge conflicts)* and confusion.

    different strategies, workflows, etc.

Merging

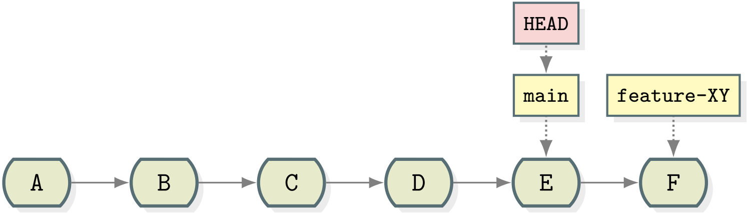
- We are done with implementing `feature-XY`



Merging

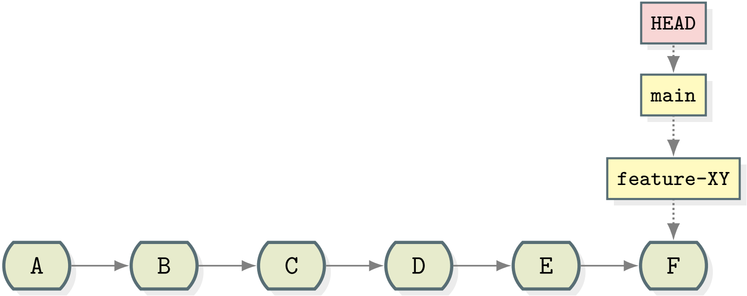
- We are done with implementing `feature-XY`
-

```
$> git switch main
```



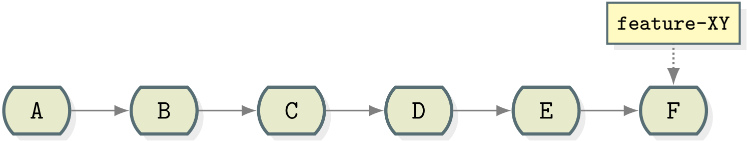
Merging

- We are done with implementing `feature-XY`
- `$> git switch main`
- `$> git merge feature-XY`



Merging

- We are done with implementing `feature-XY`
- `$> git switch main`
- `$> git merge feature-XY`
- No changes to your files
- Only a change of the branch reference

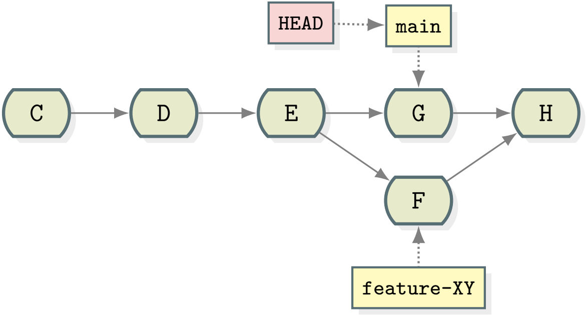


Merging - True Merge

- Actual merge of different content

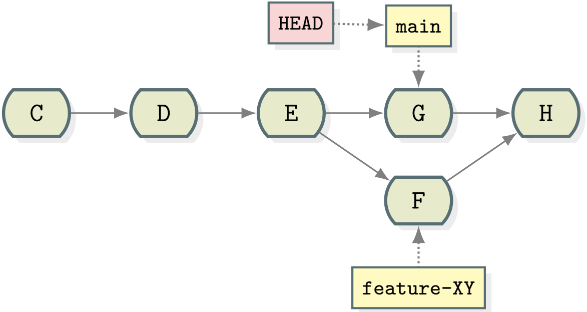
Merging - True Merge

- Actual merge of different content



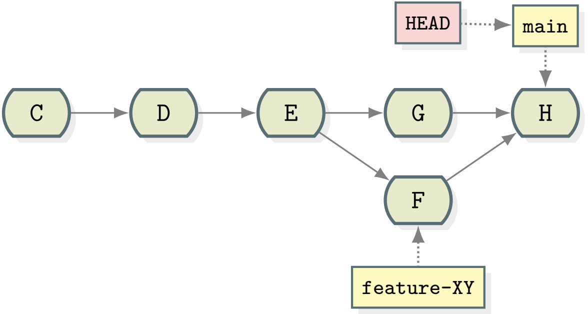
Merging - True Merge

- Actual merge of different content
- If everything runs smoothly, git will patch the new content together in a new commit



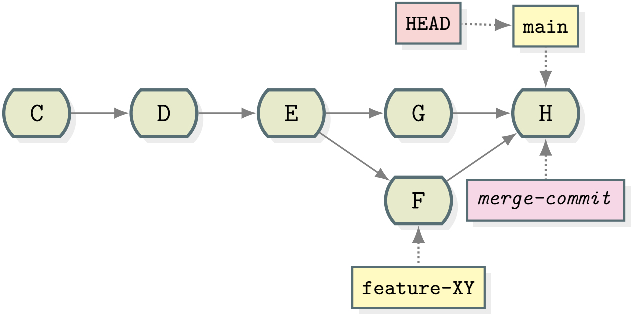
Merging - True Merge

- Actual merge of different content
- If everything runs smoothly, git will patch the new content together in a new commit



Merging - True Merge

- Actual merge of different content
- If everything runs smoothly, git will patch the new content together in a new commit



Merge Conflicts

- git branches allow you to work on different versions of the same code.

  This entails that conflicts between these versions may occur!

Merge Conflicts

- git branches allow you to work on different versions of the same code.

  This entails that conflicts between these versions may occur!
- git cannot understand the underlying logic of the changes.

  You have to resolve them yourself.

Merge Conflicts

- git branches allow you to work on different versions of the same code.
  This entails that conflicts between these versions may occur!
- git cannot understand the underlying logic of the changes.
  You have to resolve them yourself.
- In case of a three-way merge, you have three different versions:
  1. Yours
  2. Theirs
  3. The common ancestor

Merge Conflicts

```
<<<<<<< HEAD
many many more here
=======
so many conflicts here
>>>>>>> conflictBranch
```

- Conflicting hunk in a file

Remote Repositories
Branches
Working with branches
Merging and merge conflicts
Stash
Reflog
Remotes
Summary

Merge Conflicts

```
<<<<<<< HEAD
many many more here
=======
so many conflicts here
>>>>>>> conflictBranch
```

- Conflicting hunk in a file
- Shows you the changes from `HEAD` and the other branch

Merge Conflicts

- After you have resolved all conflicts: `$> git commit`

```
Merge branch 'conflictBranch'

# Conflicts:
#   foobar
#
# It looks like you may be committing a merge.
```

Stash

- Remember: git only "knows" about changes that you have commited, but git knows when you have changes, which you have not yet commited.

Stash

- Remember: git only ”knows” about changes that you have commited, but git knows when you have changes, which you have not yet commited.
- Would be overwritten on a `$> git switch <branch>`.

Stash

- Remember: git only ”knows” about changes that you have commited, but git knows when you have changes, which you have not yet commited.
- Would be overwritten on a `$> git switch <branch>`.
- `$> git stash` gives us the option to stash away changes and recover them later.

Stash

- What happens when you stash changes?
- `$> git stash` saves records of staged and unstaged changed on tracked files and reverts your working directory to `HEAD`

Stash

- What happens when you stash changes?
- `$> git stash` saves records of staged and unstaged changed on tracked files and reverts your working directory to `HEAD`
- `HEAD` is definitely in a clean state → you can switch branches!
- After you have finished working on the other branch, you can switch to the old branch and recover changes with `$> git stash apply`

Stash

- What happens when you stash changes?
- `$> git stash` saves records of staged and unstaged changed on tracked files and reverts your working directory to `HEAD`
- `HEAD` is definitely in a clean state → you can switch branches!
- After you have finished working on the other branch, you can switch to the old branch and recover changes with `$> git stash apply`
- This will recover the changes stashed last.

Multiple Stashes

- `$> git stash` and `$> git stash apply` are not limited to one stash!

Multiple Stashes

- `$> git stash` and `$> git stash apply` are not limited to one stash!
- The stash works a bit like a stack (**F**irst **I**n - **F**irst **O**ut)
- You can recover any saved stash by listing them: `$> git stash list` and passing the stash number: `$> git stash apply stash@{n}`

Stash

- You can also spawn a branch from a stash via `$> git stash branch <branchname> stash@{n}`.
- Save the stash via `$> git stash -u` to also include untracked files!

Stash

- You can also spawn a branch from a stash via `$> git stash branch <branchname> stash@{n}`.
- Save the stash via `$> git stash -u` to also include untracked files!
- Very useful if you have accidentally started working on the wrong branch!

Reflog

- Tracks whenever any of your local references change.
- Commits, checkouts, merges, etc.

Reflog

- Tracks whenever any of your local references change.
- Commits, checkouts, merges, etc.
- Can be seen as the undo-history of your local repository.

  → Even if you `$> git reset` to an older commit, you can still undo this with the reflog.

Remote Repositories

- Versions of your project hosted somewhere on the Internet, network or elsewhere.

Remote Repositories

- Versions of your project hosted somewhere on the Internet, network or elsewhere.
- This is where `$> git pull` from and `$> git push` to!

Remotes

- If you have `$> git clone`d your first repository, git added `origin` as your first remote.

Remotes

- If you have `$> git clone`d your first repository, git added `origin` as your first remote.
- `$> git remote -v` gives you a list of remotes.

Remotes

What happens when you `$> git push` / `$> git pull` ?

Remotes

What happens when you `$> git push` / `$> git pull` ?

1. As soon as your work is in a state in which you want to share it with other you push it via `$> git push <remote> <branch>`

Remotes

What happens when you `$> git push` / `$> git pull` ?

1. As soon as your work is in a state in which you want to share it with other you push it via

   ```
   $> git push <remote> <branch>
   ```

   - This will fail if there are changes on the remote branch which you have not incorporated yet → You would cause a merge conflict on the remote server!

Remotes

What happens when you `$> git push` / `$> git pull`?

Remotes

What happens when you `$> git push` / `$> git pull`?

1. You can get all new changes and branches from the remote via `$> git fetch <remote>`

Remotes

What happens when you `$> git push` / `$> git pull` ?

1. You can get all new changes and branches from the remote via `$> git fetch <remote>`
   - This only downloads those changes, no direct changes to your files!

Remotes

What happens when you `$> git push` / `$> git pull`?

1. You can get all new changes and branches from the remote via `$> git fetch <remote>`
   - This only downloads those changes, no direct changes to your files!
   - You can then `$> git merge <remote>/<branch>` the changes into your local branch.
   - Usually you just want to get changes from the remote main, or any other branch, and merge them. `$> git pull remote main` is doing exactly that.

Remotes

- If you want to work with multiple remotes you can add them via

```
$> git remote add <shortname> <git-url>
```

- This lets you push your work to different remote repositories!

Remotes

- If you want to work with multiple remotes you can add them via

```
$> git remote add <shortname> <git-url>
```

- This lets you push your work to different remote repositories!
- Can be useful if you want to work both with a private and a public repository.

Remotes

- If you want to work with multiple remotes you can add them via

  ```
  $> git remote add <shortname> <git-url>
  ```

- This lets you push your work to different remote repositories!
- Can be useful if you want to work both with a private and a public repository.
- `$> git remote show <shortname>` gives you useful information about the remote.

Summary

- What is a non-linear workflow?
- What is a branch?
- What is a git reference?
- What does a merge do?
- What is a merge-conflict?
- What is a stash?
- What is the reflog?
- What are remotes?