

# 19 – Developing in branches with Git

Bálint Aradi

**Scientific Programming in Python (2026)**

<https://atticlectures.net/scipro/python-2026/>



# Branch & merge in one repository (#1)

## Creating repository

```
mkdir -p gitdemo/hello  
cd !$  
git init
```

Last argument (\$) of last command (!)

```
git add hello.py  
git ci -m "Initial checkin"
```

```
print("Hello!") hello.py
```

● **main** Initial checkin

## Creating branch **cleanup**

```
git branch cleanup
```

● **cleanup** — **main** Initial checkin

## Switching to branch **cleanup**

```
git switch cleanup
```

“cleanup” & “main” point to same commit

● **cleanup** — **main** Initial checkin

## Checking current branch

```
git branch
```

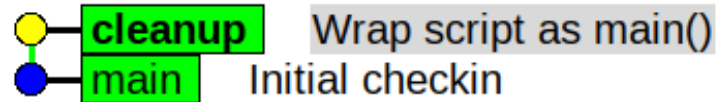
\* cleanup  
main

All branches, current one marked with “\*”

# Branch & merge in one repository (#2)

## Developing on branch **cleanup**

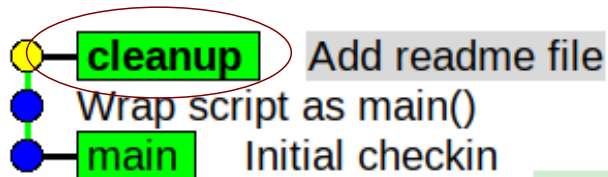
```
git add -u
git commit -m "Wrap script as main()"
```



Pointer "cleanup" (actual branch) advanced, "main" remains.

Create README.rst

```
git add README.rst
git commit -m "Add readme file"
```



```
def main():                                hello.py
    print("Hello!")

if __name__ == "__main__":
    main()
```

```
*****                                README.rst
Hello
*****

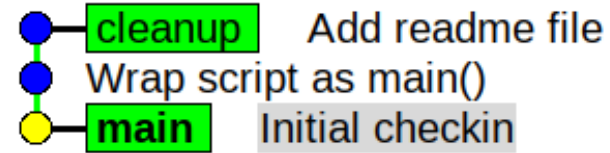
Trivial greeting project to
demonstrate the usage of
multiple git branches.
```

**Branch name** = Named pointer pointing to a given commit representing the end of a named development (time)line

# Branch & merge in one repository (#3)

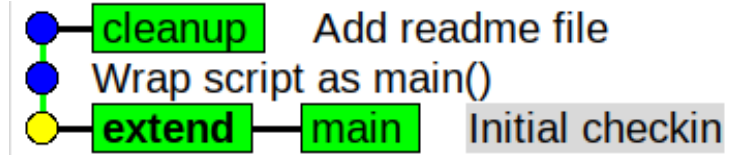
Switching back to **main** branch

```
git switch main
```



Creating a new branch **extend** starting from the state of the project on “main”

```
git switch -c extend
```



→ Content of `hello.py` changed back to the state as in the **main** branch:

```
print("Hello!")      hello.py
```

→ File `README.rst` does not exist (it only exists in the **cleanup** branch, but not in **main**)

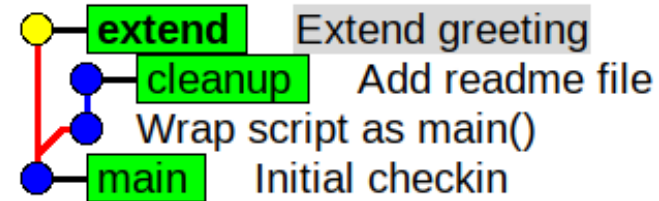
# Branch & merge in one repository (#4)

## Developing on branch "extend"

```
print("Hello, World!") hello.py
```

Change file content

```
git add -u  
git commit -m "Extend greeting"
```



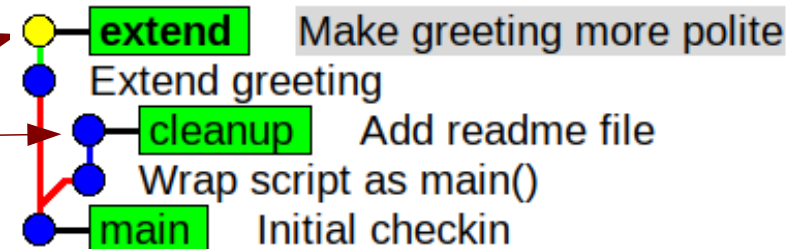
## Developing on branch "extend"

```
print("Hello, World!") hello.py  
print("How are you doing?")
```

Change file content

```
git add -u  
git commit -m "Make greeting more polite"
```

Branches **extend** and **cleanup** diverged



# Branch & merge in one repository (#5)

Merging changes from first branch to **main** branch

```
git switch main
```

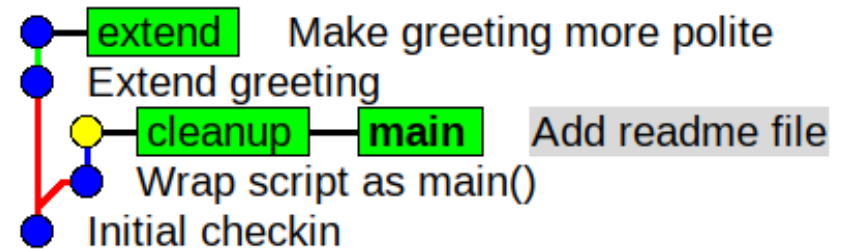
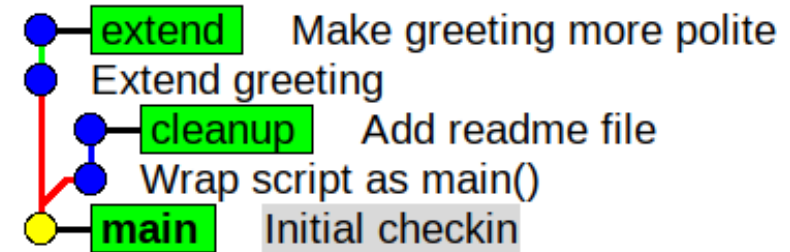
```
git merge cleanup
```

Updating b97c415..d66bbe7

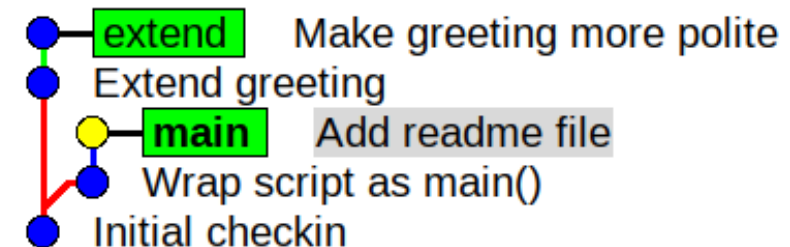
**Fast-forward**

```
git branch -d cleanup
```

Deleted branch cleanup ...



Commit pointed by “cleanup” can be reached from commit pointed by “main” by going only forward in time: Pointer “main” had been simply forwarded to point to “cleanup” (**fast forward**)



Deleting unnecessary **pointer** (not the commit) “cleanup”

(all commits until “cleanup” are contained in the history of the commit pointed by “main”)

# Branch & merge in one repository (#6)

## Merging changes from second branch to main project

```
git switch main  
git merge extend
```

Just to make sure we are on the main branch

**Auto-merging** hello.py

**CONFLICT** (content): Merge conflict in  
hello.py

Automatic merge failed; fix conflicts  
and then commit the result.

- The **same lines** have been **changed** on main (due to merge of branch “cleanup”) and on branch “extend”
- Git can not apply both changes simultaneously
- **Conflict(s)** must be solved manually
- **Conflict(s)** are specially marked in the file

```
<<<<<<<< HEAD                                     hello.py  
def main():  
    print("Hello!")  
  
if __name__ == "__main__":  
    main()  
  
=====  
print("Hello, World!")  
print("How are you doing?")  
>>>>>>> extend
```

# Branch & merge in one repository (#7)

## Fix merge conflicts and commit merge

```
def main():  
    print("Hello, World!")  
    print("How are you doing?")
```

hello.py  
(resolved version)

```
if __name__ == "__main__":  
    main()
```

```
git add hello.py  
git commit
```

Commits merge  
(= changes from merged branch  
+ manual changes for conflict resolution)

Tells git that conflict has  
been manually resolved

<<<<<<< HEAD

hello.py

```
def main():  
    print("Hello!")
```

Conflicting  
change on  
current  
(main) branch

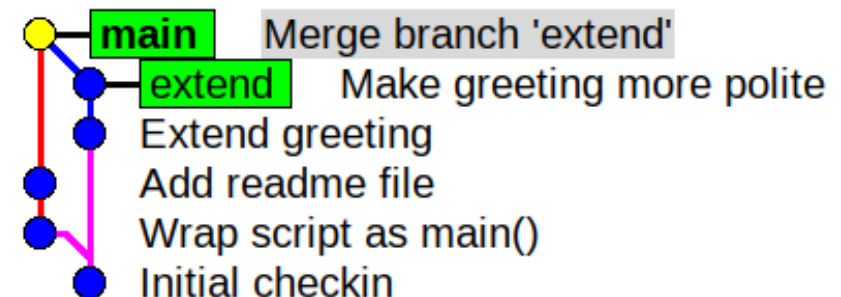
```
if __name__ == "__main__":  
    main()
```

=====

```
print("Hello, World!")  
print("How are you doing?")
```

Conflicting  
change on  
branch being  
merged  
(extend)

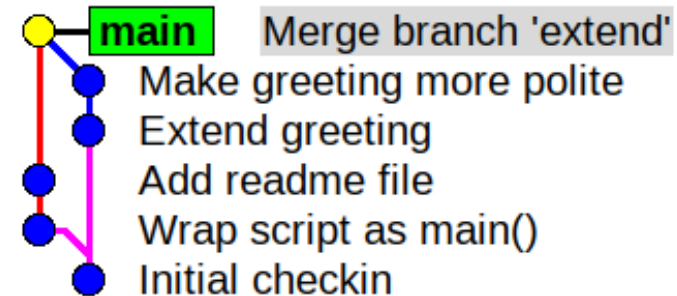
>>>>>>> extend



## Branch & merge in one repository (#8)

```
git branch -d extend
```

Deleted branch extend (was 779ffb1).



- Deleting superfluous pointer, since commits on “extend” has been merged into main  
→ they are part of the history of the commit where “main” points to  
(they can be reached from “main” by going only backwards in time)

Main branch contains all changes from both feature branches  
+ all changes necessary to resolve the conflicts between them

# Fast forward vs. explicit merge commit

## Advantages of fast-forward merges

- No extra merge commits in the logs
- Keeps git-history linear (some projects prefer such history...)

## Advantages of explicit merge commits

- It is clear, where the changes came from (feature branch)
- Feature can be easily removed (by removing/reverting) a single merge commit

# Forcing merge commits

- The `--no-ff` option can enforce an explicit merge commit, even when fast forward were possible

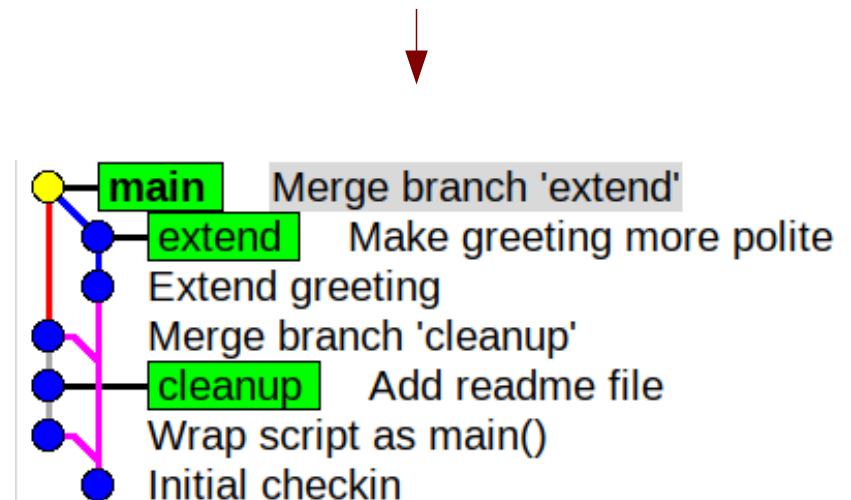
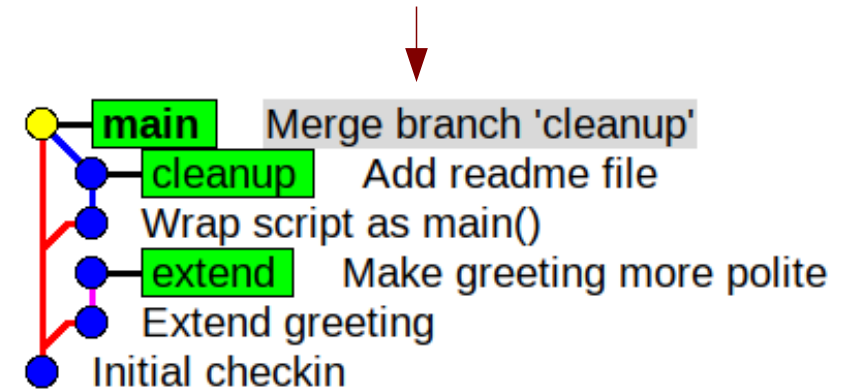
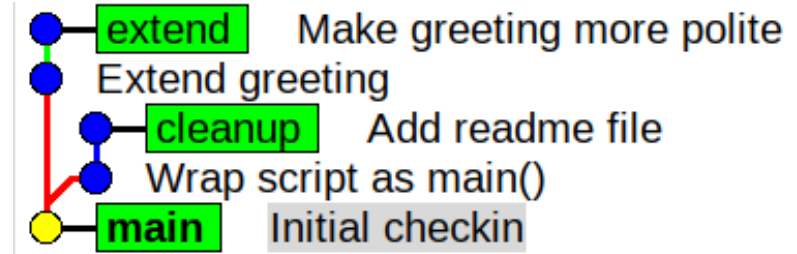
```
git merge --no-ff cleanup
```

Fast forward not possible here, so git would automatically make merge commit here, but option can still be set.

```
git merge --no-ff extend
```

```
CONFLICT (content): Merge conflict in  
hello.py
```

```
git add hello.py  
git commit
```



# Mainpulating conflicts in IDEs

- Most IDEs allow the manipulation of files with conflict markers:

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 | <<<<<<< HEAD (Current Change)
2 | def main():
3 |     print("Hello!")
4 |
5 | if __name__ == "__main__":
6 |     main()
7 | =====
8 | print("Hello, world!")
9 | print("How are you doing?")
10 | >>>>>> extend (Incoming Change)
```