19 – Developing in branches with Git

Bálint Aradi

Scientific Programming in Python (2025)

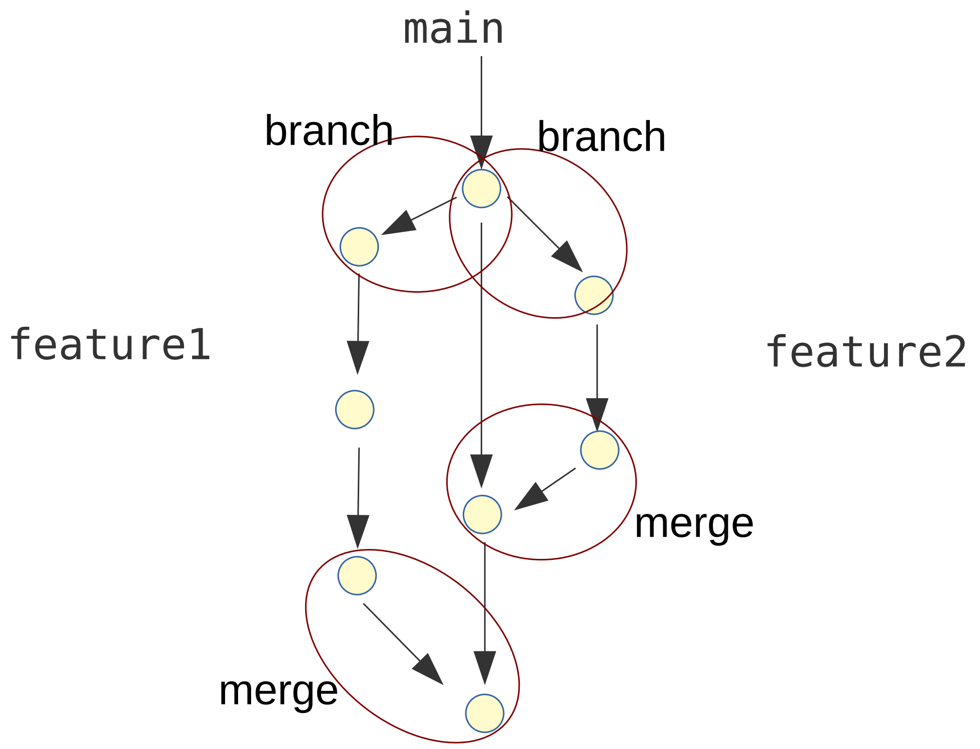
https://atticlectures.net/scipro/python-2025/

Branch & merge in one repository

Parallel development of features:

- Multiple **independent features** are explored at the same time
- A **bug** has to be fixed in an **older version** of the code (e.g. last release) without exposing unmature/unfinished new features

Typical workflow



- Features are implemented in **branches** (independent development histories)
- Branches start from the actual state of the main project
- **Every** new feature / significant **change** gets its **own branch**
- If implementation finished, changes are added (merged) to main project
- **Conflicting changes** in parallel branches (e.g. same lines changed), must be manually **resolved** (during merge).

Branch & merge in one repository (#1)

Creating repository

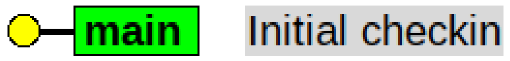
```
mkdir -p gitdemo/hello
cd !$
git init
```

Last argument ($)
of last command (!)

```
print("Hello!")
```

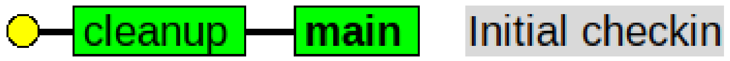
hello.py

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



Creating branch **cleanup**

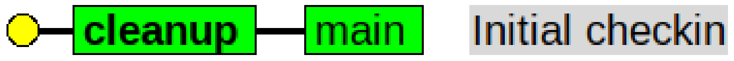
```
git branch cleanup
```



“cleanup” & “main” point to same commit

Switching to branch **cleanup**

```
git switch cleanup
```



Checking current branch

```
git branch
```

```
* cleanup
  main
```

All branches, current one marked with “*”

Branch & merge in one repository (#2)

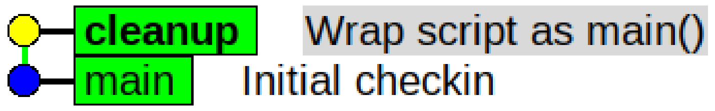
Developing on branch **cleanup**

hello.py

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

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

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



Pointer “cleanup” (actual branch) advanced, “main” remains.

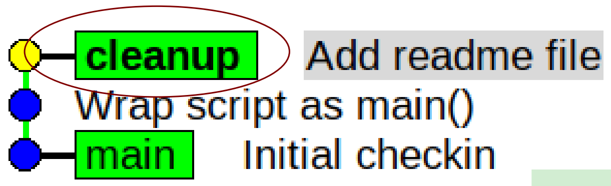
Create README.rst

README.rst

```
*****
Hello
*****


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

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

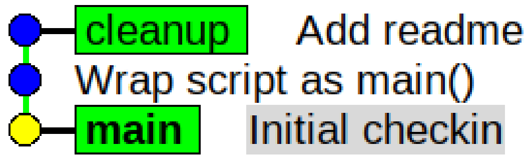


Branch name = Named pointer pointing to a given commit representing the end of a named development (time)line

Branch & merge in one repository (#3)

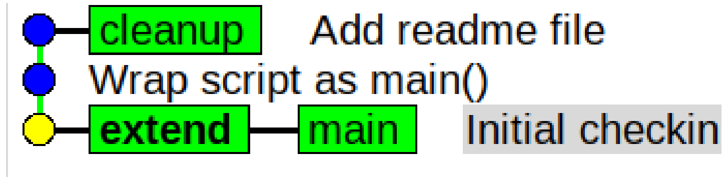
Switching back to **main** branch

```
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)

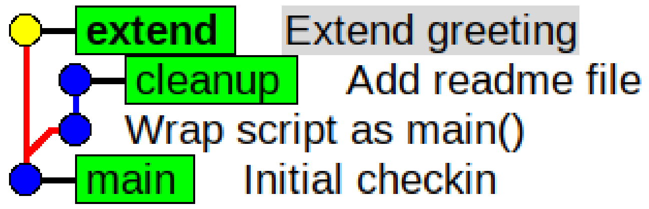
Developing on branch “extend”

Change file content

```
print("Hello, World!")
```

hello.py

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



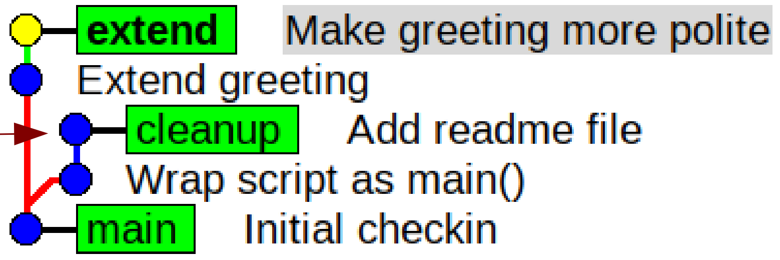
Developing on branch “extend”

Change file content

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

hello.py

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

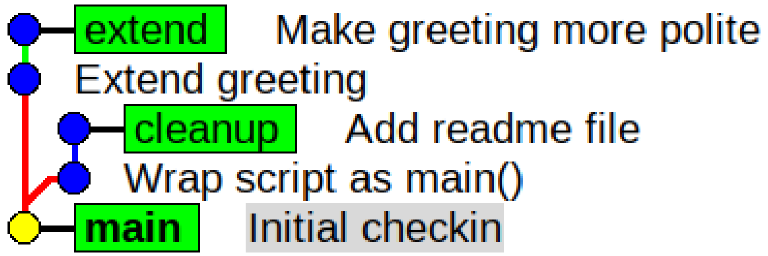


Branches **extend** and **cleanup** diverged

Branch & merge in one repository (#5)

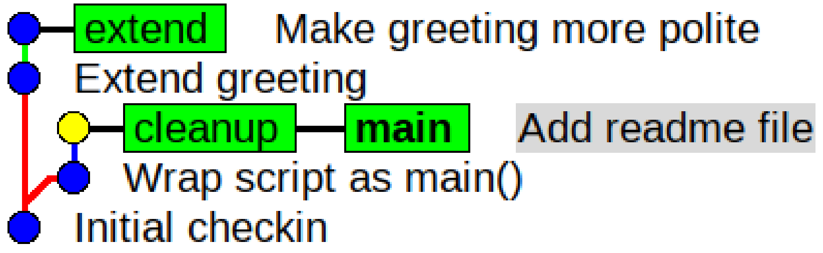
Merging changes from first branch to **main** branch

```
git switch main
```



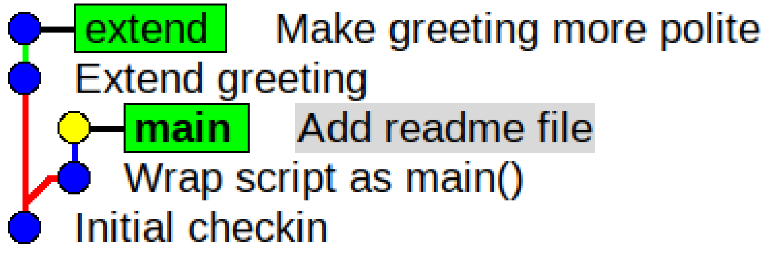
```
git merge cleanup
```

```
Updating b97c415..d66bbe7
```
`Fast-forward`



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**)

```
git branch -d cleanup
```



```
Deleted branch cleanup ...
```

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.
```

hello.py

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

if __name__ == "__main__":
    main()
=======
print("Hello, World!")
print("How are you doing?")
>>>>>>> extend
```

- 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

Branch & merge in one repository (#7)

Fix merge conflicts and commit merge

hello.py
(resolved version)

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

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

hello.py

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

if __name__ == "__main__":
    main()
=======
print("Hello, World!")
print("How are you doing?")
>>>>>>> extend
```

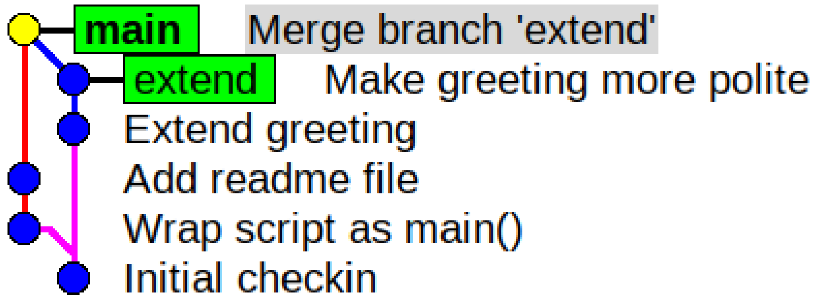
Conflicting change on current (main) branch

Conflicting change on branch being merged (extend)

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

Tells git that conflict has been manually resolved

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

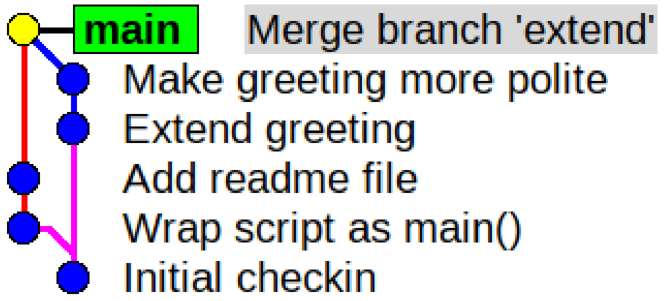


```
main    Merge branch 'extend'
extend  Make greeting more polite
Extend greeting
Add readme file
Wrap script as main()
Initial checkin
```

Branch & merge in one repository (#8)

```
git branch -d extend
```

```
Deleted branch extend (was 779ffb1).
```



- Deleting superfluos 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

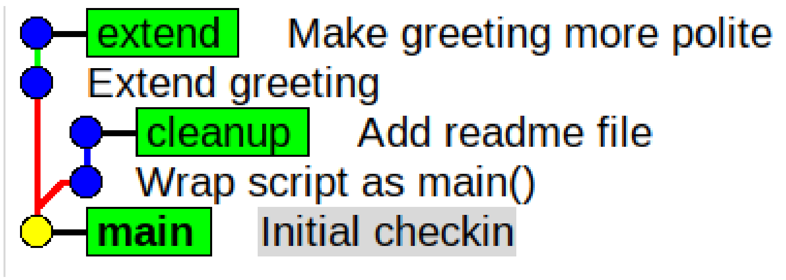
```
git merge --no-ff cleanup
```

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

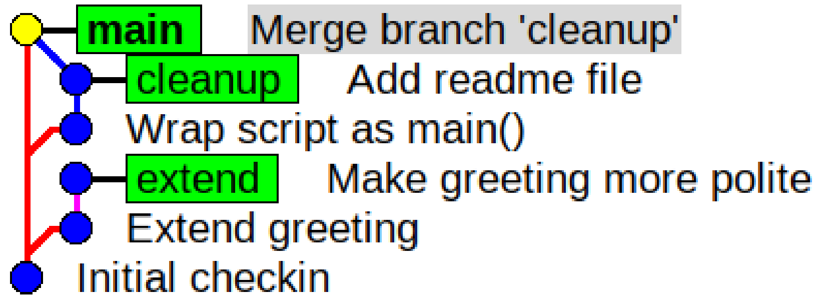
```
git merge --no-ff extend
```

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

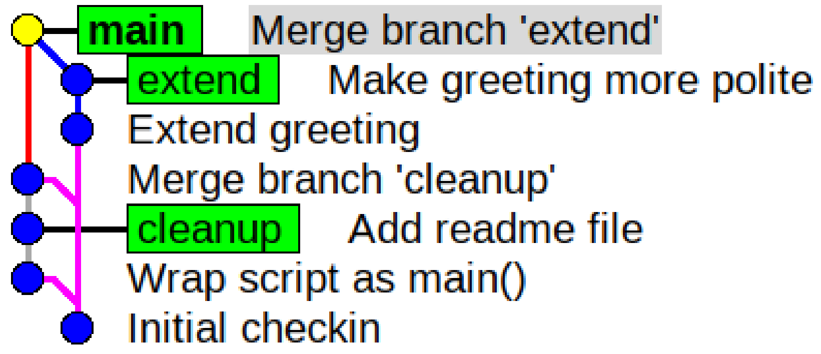
```
git add hello.py
git commit
```



extend Make greeting more polite
Extend greeting
cleanup Add readme file
Wrap script as main()
main Initial checkin



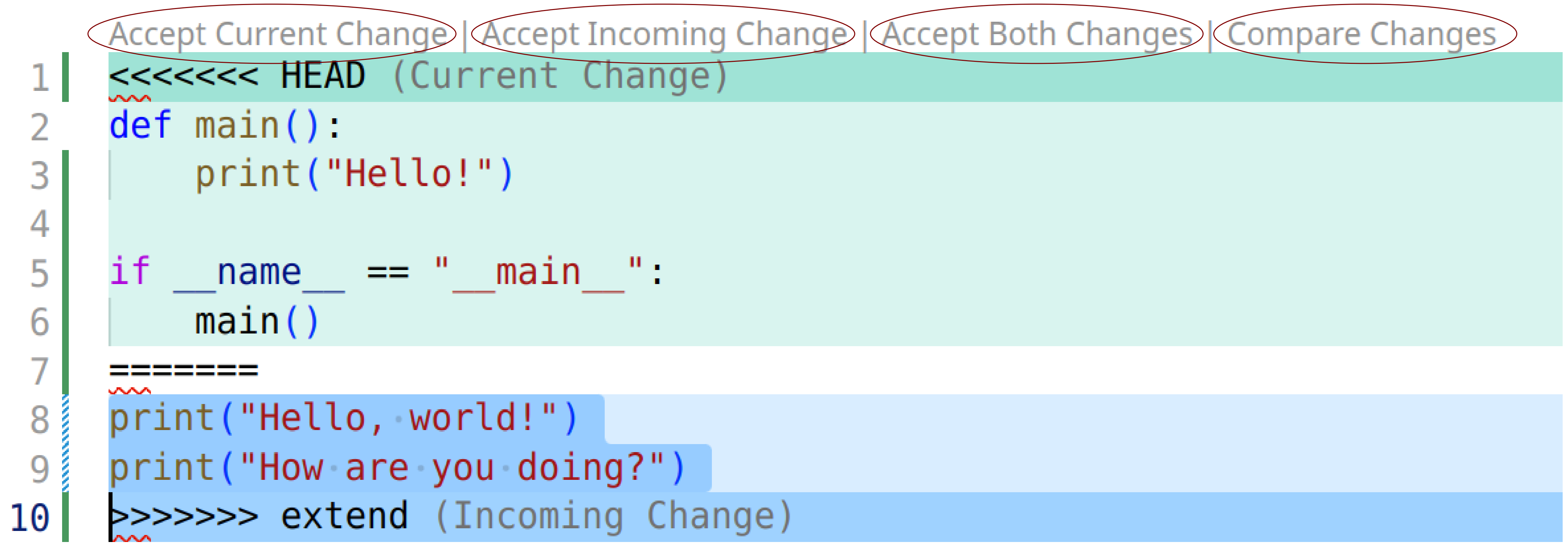
main Merge branch 'cleanup'
cleanup Add readme file
Wrap script as main()
extend Make greeting more polite
Extend greeting
Initial checkin



main Merge branch 'extend'
extend Make greeting more polite
Extend greeting
Merge branch 'cleanup'
cleanup Add readme file
Wrap script as main()
Initial checkin

Mainpulating conflicts in IDEs

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



Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<<< HEAD (Current Change)
def main():
    print("Hello!")

if __name__ == "__main__":
    main()
=======
print("Hello, world!")
print("How are you doing?")
>>>>>>> extend (Incoming Change)
```