git

An introduction

ÉCOLE **NORMALE**
SUPÉRIEURE
DE **LYON**

Ali Farnudi, Fall 2021

Without a **Version Control System**

- my_thesis_1.tex
- my_thesis_2.tex
- my_thesis_3.tex
- my_thesis_3b.tex
- my_thesis_4.tex
- my_thesis_5.tex
- my_thesis_6_first_version_showed_prof.tex
- my_thesis_7_corrections.tex
- my_thesis_8_semifinal.tex
- my_thesis_9_final.tex
- my_thesis_10_final_2.tex
- my_thesis_11_final_final.tex
- ....

Without a **Version Control System**

- my_thesis_1.tex
- my_thesis_2.tex
- my_thesis_3.tex
- my_thesis_3b.tex
- my_thesis_4.tex
- my_thesis_5.tex
- my_thesis_6_first_version_showed_prof.tex
- my_thesis_7_corrections.tex
- my_thesis_8_semifinal.tex
- my_thesis_9_final.tex
- my_thesis_10_final_2.tex
- my_thesis_11_final_final.tex
- ....

Loads of emails
confusion

Without a **Version Control System**

- my_code_1.py
- my_code_2.py
- my_code_2a.py
- my_code_2b.py
- my_code_2c.py
- my_code_2d.py
- my_code_3_with_Ralfs_suggestion.py
- my_code_4.py
- my_code_5_bug_found.py
- my_code_6_bug_fixed.py

Without a **Version Control System**

- my_code_1.py
- my_code_2.py
- my_code_2a.py
- my_code_2b.py
- my_code_2c.py
- my_code_2d.py
- my_code_3_with_Ralfs_suggestion.py
- my_code_4.py
- my_code_5_bug_found.py
- my_code_6_bug_fixed.py

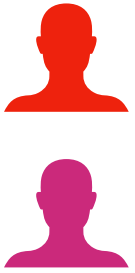
This bug appeared when I did **that** in **some** version...

I want to try something in the version before the bug was introduced

I want to keep many versions of different files

Without a **Version Control System**

- my_code_1.py
- my_code_2.py
- my_code_2a.py
- my_code_2b.py
- my_code_2c.py
- my_code_2d.py
- my_code_3_with_Ralfs_suggestion.py
- my_code_4.py
- my_code_5_bug_found.py
- my_code_6_bug_fixed.py

\+  =

100 bugfix/day

Loads of email exchanges

People go crazy

Code doesn't work

This bug appeared when I did **that** in **some** version...

I want to try something in the version before the bug was introduced

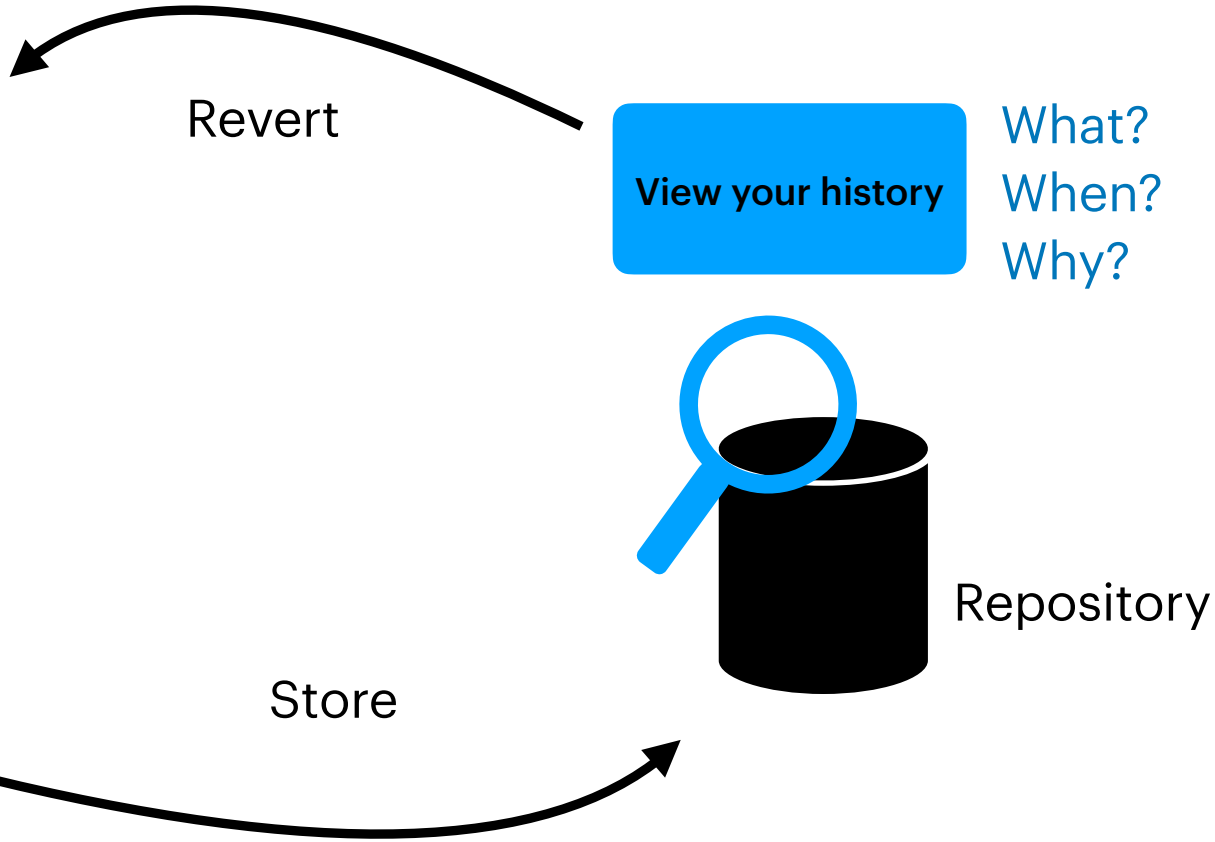
I want to keep many versions of different files

git is the most popular VCS

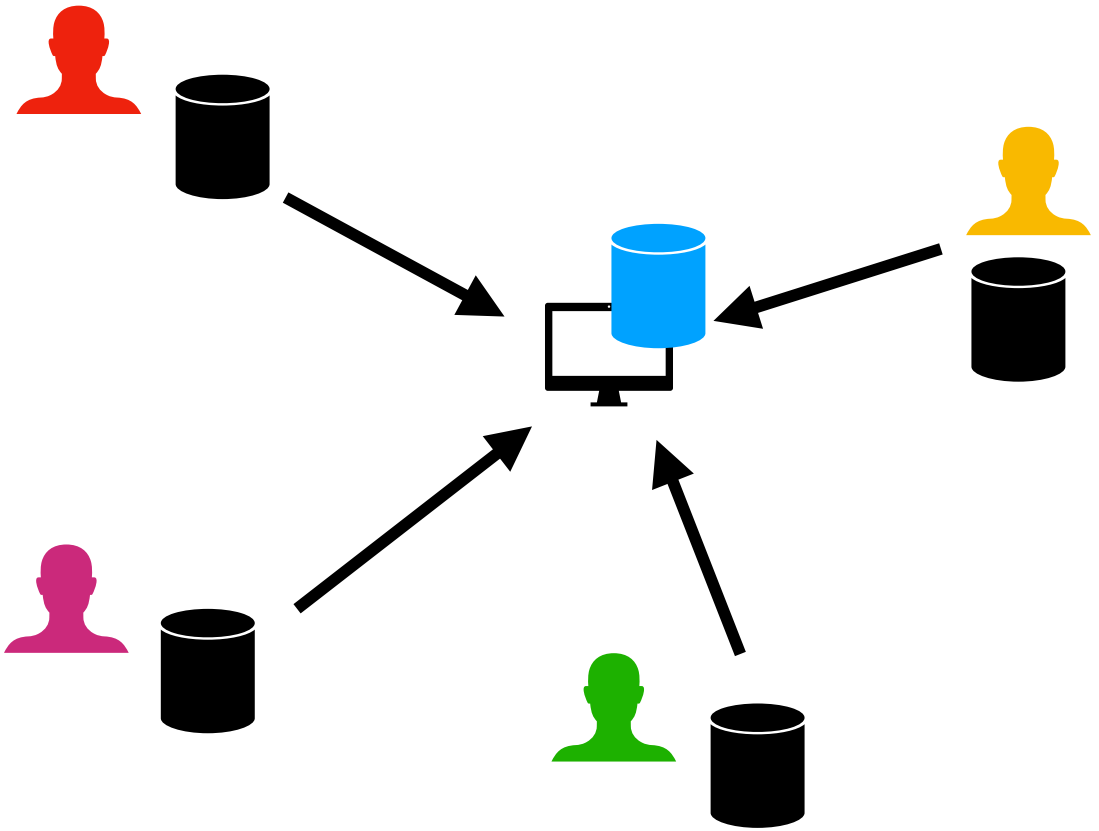
```
def getColourIndex(r):
    if r==100:
        return 0
    elif r == 1000:
        return 1
    elif r == 10_000:
        return 2
    else:
        return 3

def getPlotStat(tempStat, Teff,tempThreshold):
    if tempStat=='all':
        plotStat=True
    elif tempStat=='high':
        if Teff>tempThreshold:
            plotStat=True
        else:
            plotStat=False
    elif tempStat=='low':
        if Teff<tempThreshold:
            plotStat=True
        else:
            plotStat=False
    return plotStat
def main():

    df = loadData()
    df = df.sort_values(by=['R'])

    from matplotlib.pyplot import cm
    color = cm.rainbow(np.linspace(0,1,len(df.index)))
    radiuslist = [500,1000,5000,10000,50000,100000]
    color = cm.rainbow(np.linspace(0,1,len(radiuslist)))
    alpha= 1
    for c in color:
```

Records snapshots

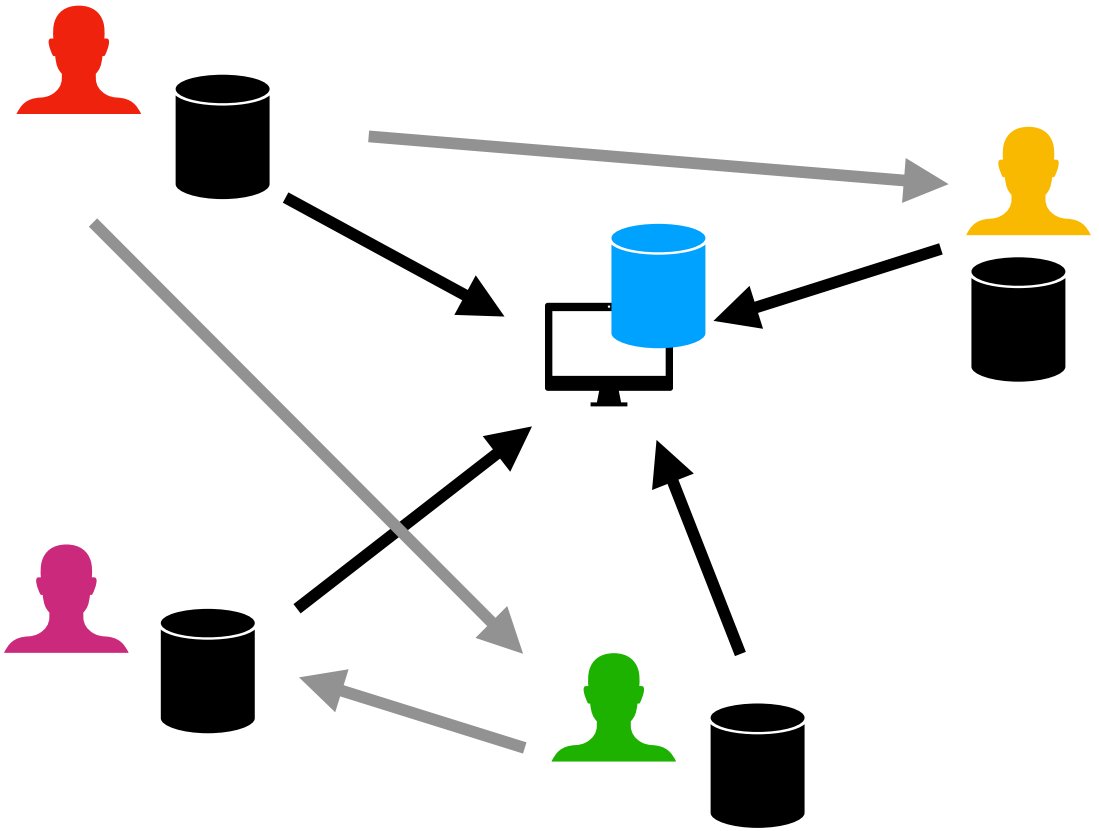
Store



git

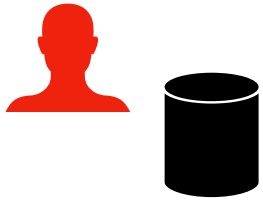


git



git

Solo

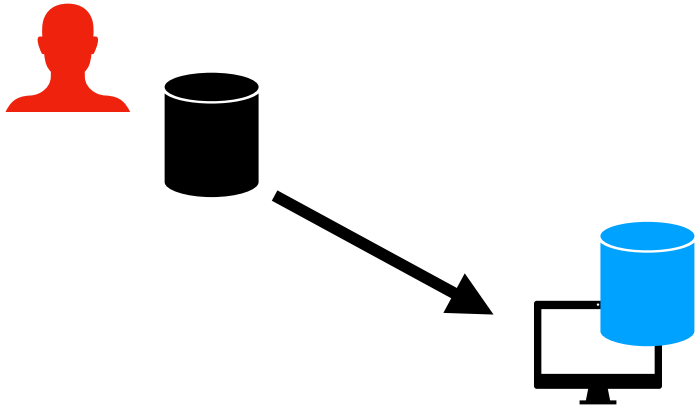


- Code in a repository
- Track all past versions + rollback
- Compare past versions
- Branch development

git

Solo

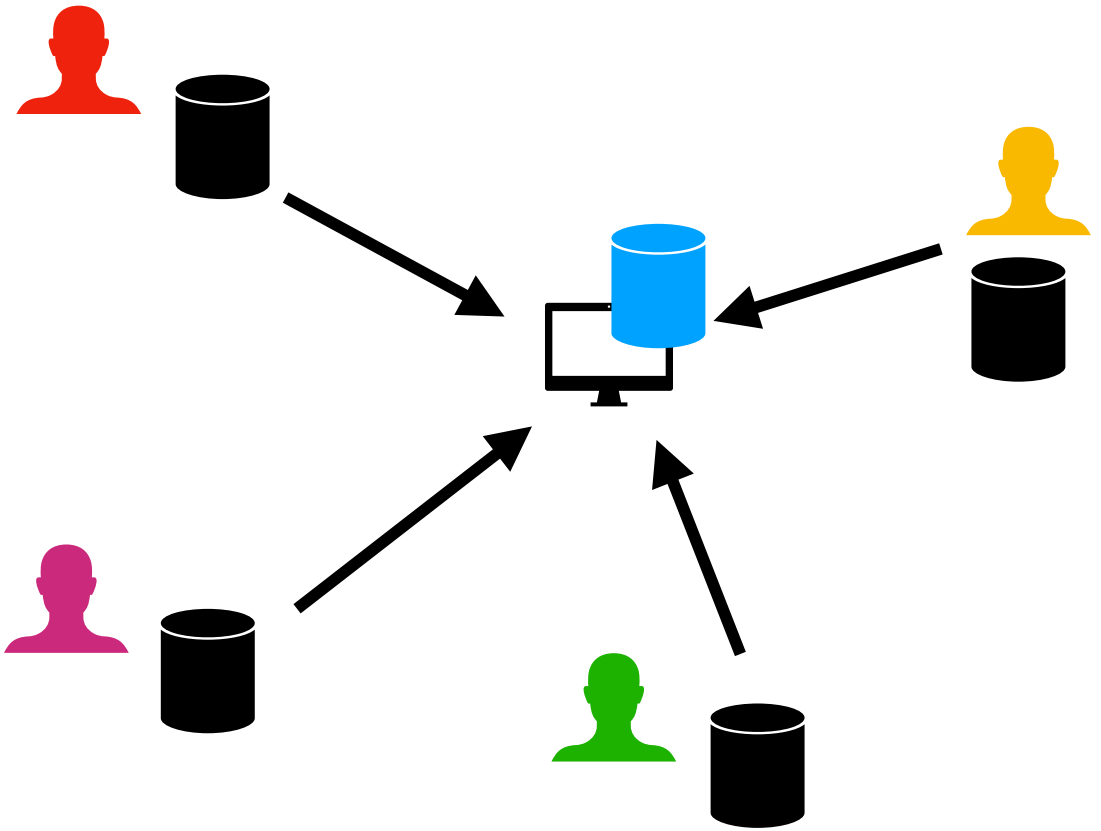
- Code in a repository
- Track all past versions + rollback
- Compare past versions
- Branch development



git

Solo

- Code in a repository
- Track all past versions + rollback
- Compare past versions
- Branch development



Collaboration

- Common remote repository
- Merge contributions of different developers
- See **who**, **when** wrote **what**, and **why**

Install Git

Git website

- Mac
- Linux
- Windows
  - Git BASH

Using Git

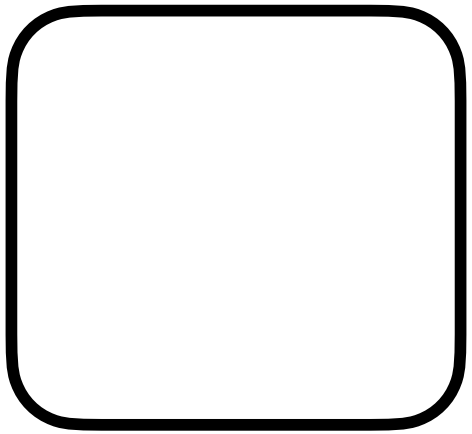
- Command line
- IDEs + code editors
  - Xcode (MacOS)
  - Visual Studio (MS Windows)
- GUIs
  - Tower (free for students)
  - Git kraken
  - Sourcetree (Mac and Windows)
  - More on the git website

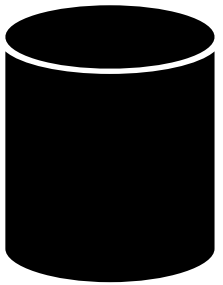
Hands-on config

- Configure your git settings:
  - $ git config --global user.name "[name]"
  - $ git config --global user.email "[email address]"
  - $ git config --global color.ui auto
  - $ git config --global core.editor “editor name”
  - $ git config --global -e
  - $ git config -h
  - $ git config --help

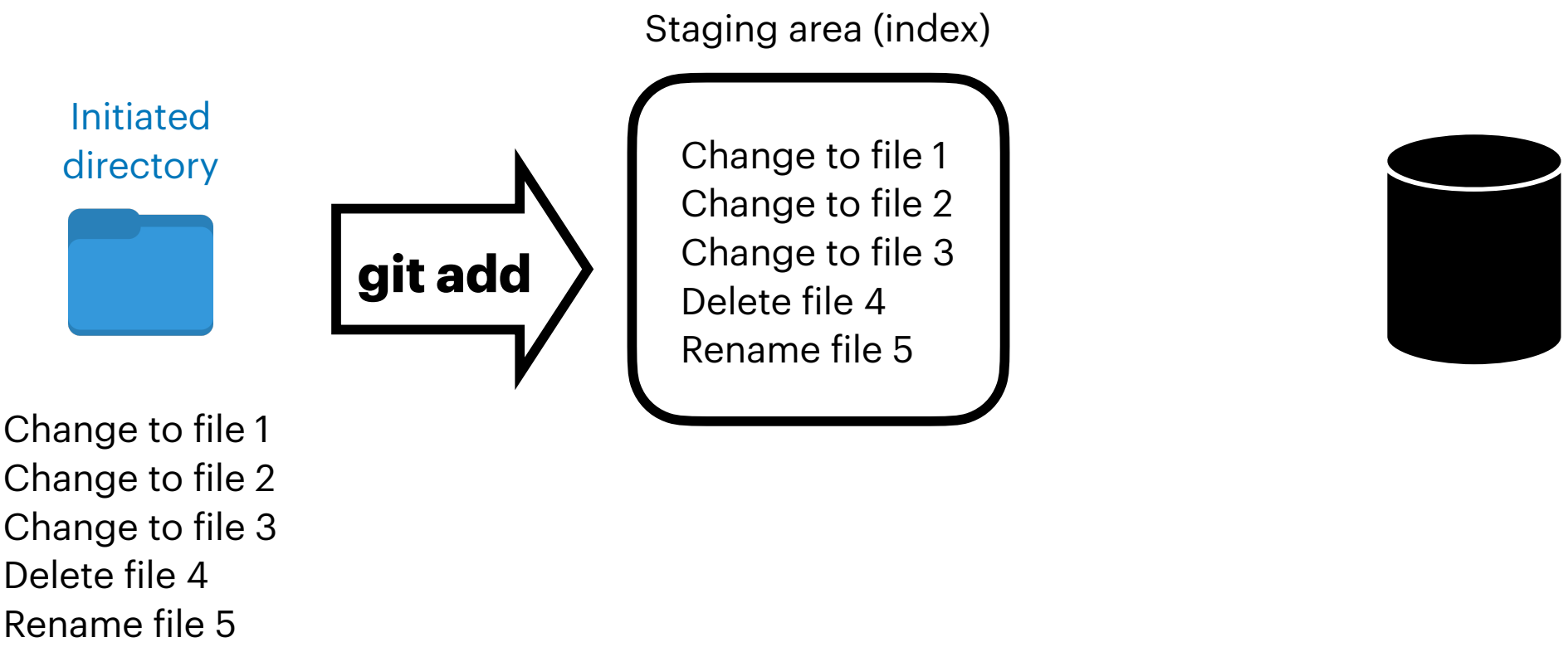
Initiated directory

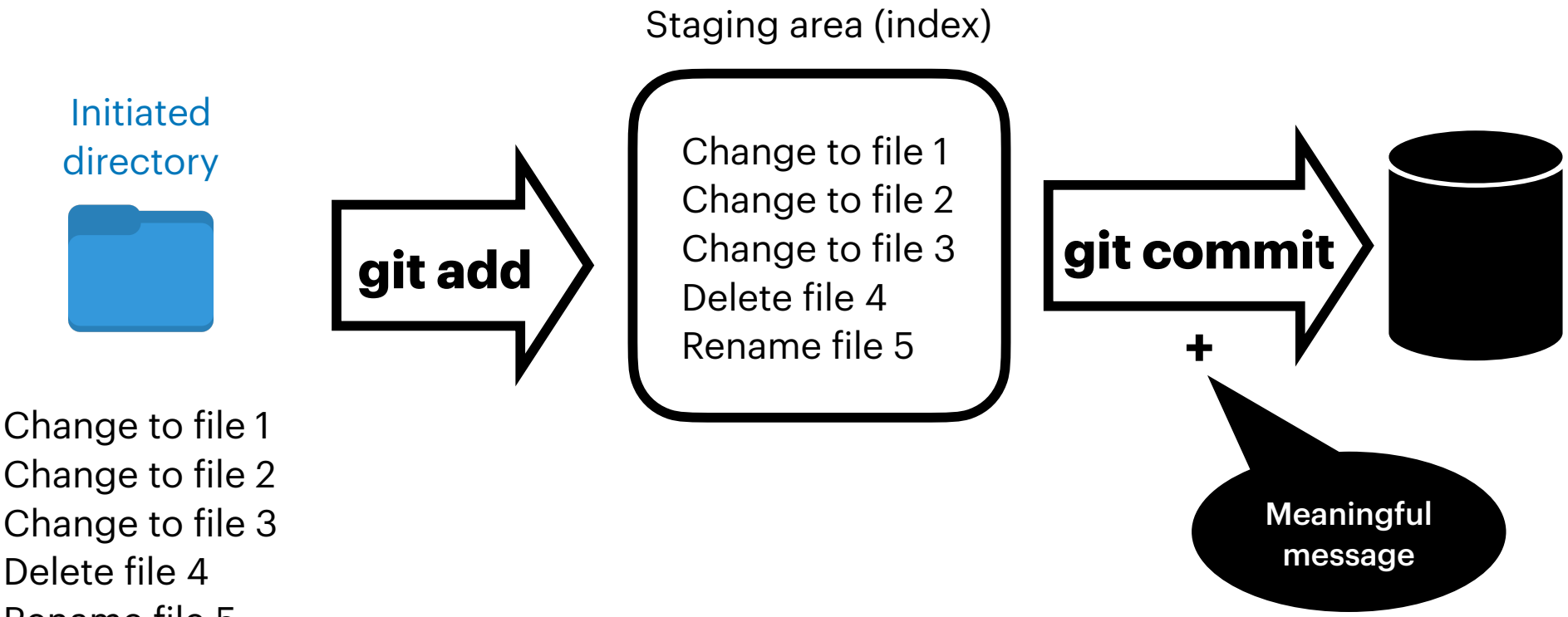
Staging area (index)

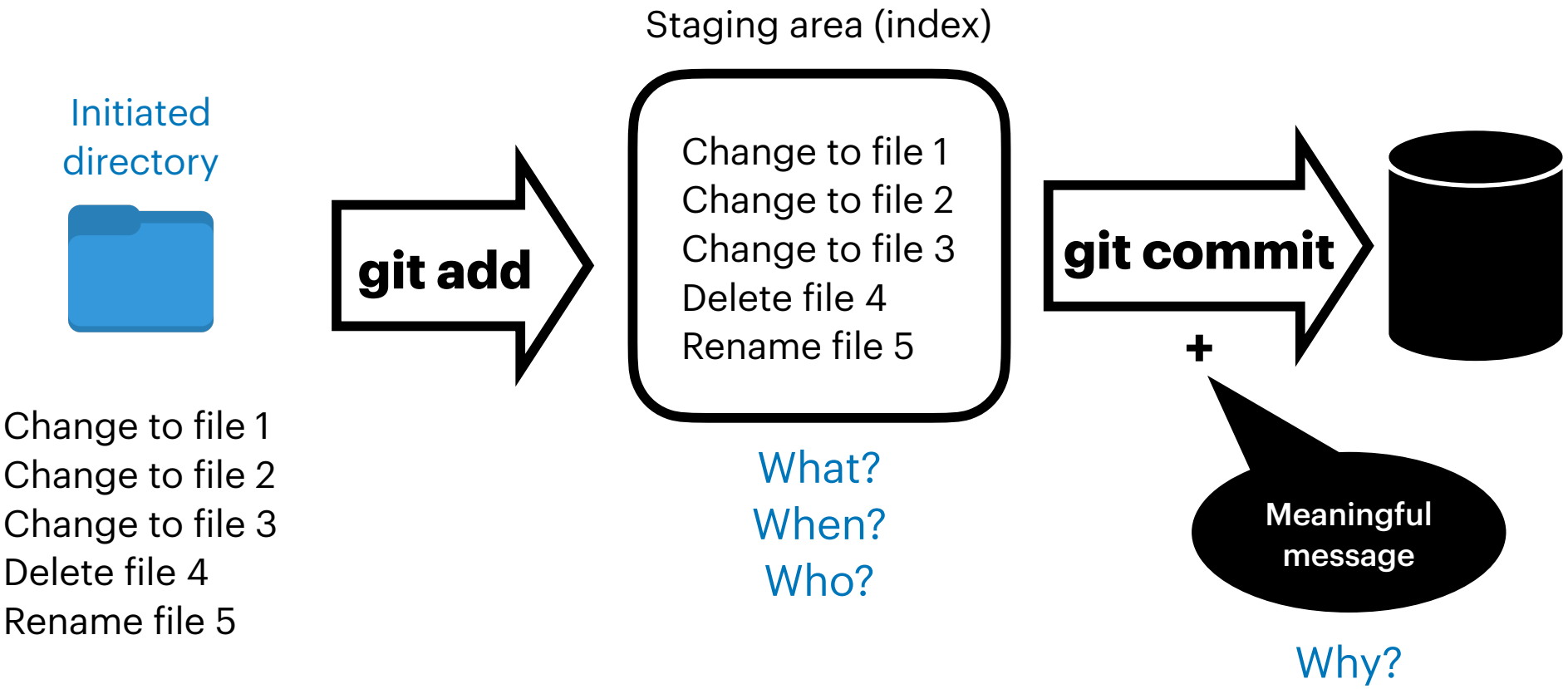




Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5


Staging area (index)
Initiated directory
git add
Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5
Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5


Staging area (index)
Initiated directory
git add
Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5
git commit
+
Meaningful message
Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5


Staging area (index)
Initiated directory
git add
Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5
git commit
+
Meaningful message
Change to file 1
Change to file 2
Change to file 3
Delete file 4
Rename file 5
What?
When?
Who?
Why?

Hands-on init+commit

- Initiate git in a directory:
  - $ git init
- Make 3 text files:
  - $ git status
  - $ git add
- Committing to changes:
  - $ git commit
  - $ git commit -a

- What to write in commits?
  - Message size
  - Title and and details
  - Don’t cram multiple tasks into one commit: typo, bugfix, new function

Hands-on rm mv

- Delete files
  - $ git status
  - $ git ls-files
  - $ rm file2.txt
  - $ git add file2.txt

} $ git rm file2.txt

$ git commit -m “”

- Rename files
  - $ mv file3.txt main.cpp
  - $ git add file1.txt
  - $ git add main.cpp

} $ git mv file3.txt main.cpp

Hands-on ignore

- Create bin/app.out
- Ignore the files
  - Create a .gitignore file
  - $ git add .gitignore
    - : *.DS_Store *.log *.aux
  - Modify bin/app.out
  - $ git status
- Create bin/app2.out
- Add and commit
- Add bin/ to .gitignore
- Modify bin/app2.out
  - $ git status
  - $ git ls-files
  - $ git rm -h
  - $ git rm --cached bin/
  - $ git rm --cached -r bin/

https://github.com/github/gitignore

Hands-on alt status diff

- Try:
  - $ git status -s
- Add a file to the staging environment
  - $ git diff
- Modify a staged file
    - $ git diff --staged
    - $ git config --global diff.tool vimdiff

Hands-on log

- Looking at the changes
  - $ git log
  - $ git log -3
  - $ git log -p
  - $ git log --stat —summary
  - $ git log --follow [file]
  - $ git log —oneline
  - $ git log --after 2017-07-04
  - $ git log —author=“afarnudi”
  - $ git log --grep="cell"
- $ git show 1b2e1d63ff
- $ git show HEAD
- $ git show HEAD~1
- $ git show HEAD~2:file1.txt

Hands-on restore

- Restoring changes
  - $ git restore --staged file1.txt
  - $ git clean -h
- Restoring deleted files
  - $git rm file1.cpp; $ git commit
  - $ git restore -h
  - $ git restore --source=HEAD~1 file1.cpp

Hands-on diff log

- Give commits, custom names
  - $ git tag v2.5 1b2e1d63ff
  - $ git diff v2.5 HEAD
- Comparing histories
  - $ git diff b497041c642..df658276d
  - $ git diff v2.5..df658276d
  - $ git log v2.5..v2.6
  - $ git log v2.5..
  - $ git log --since="2 weeks ago”
  - $ git log v2.5.. Makefile

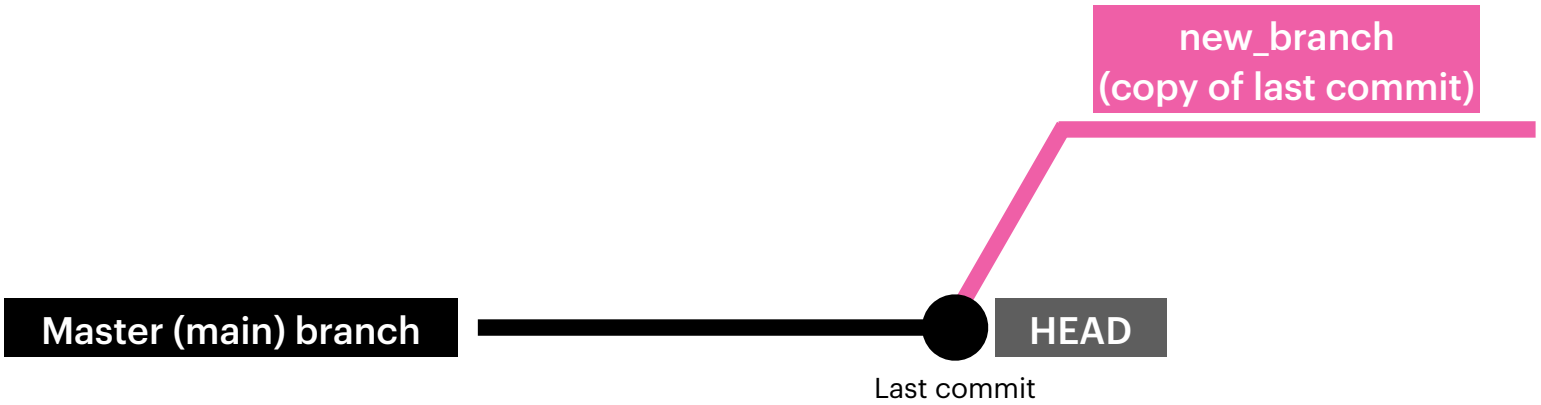
Hands-on remote

- Create a Github/Gitlab account
- $ git remote add origin https://github.com/harishrajora805/myFirstRepo.git
- Or
- $ git clone [url]
- Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

Branching

$ git branch List of branches (Master)

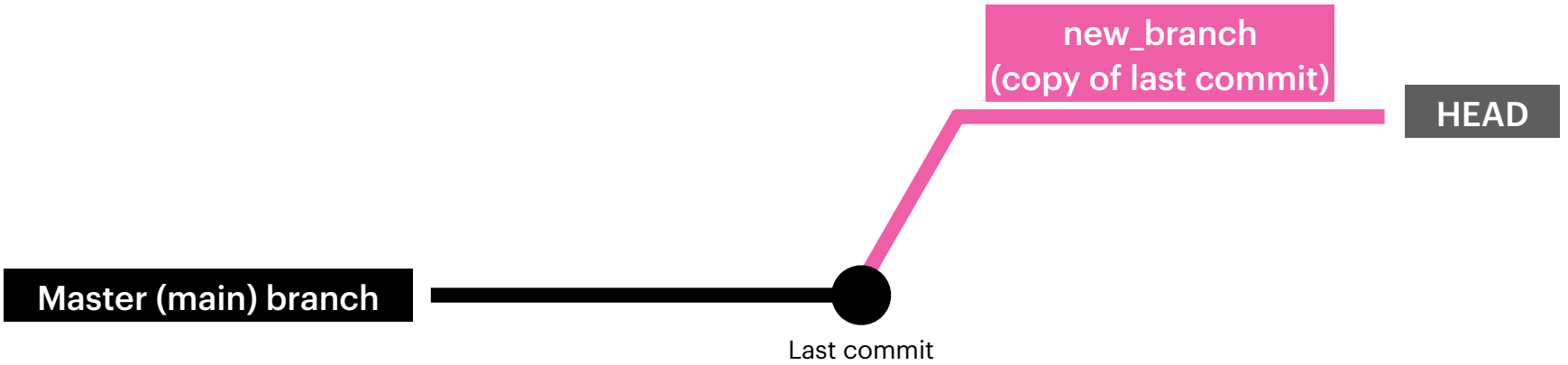
Branching



$ git branch List of branches (Master)

$ git branch new_branch

Branching

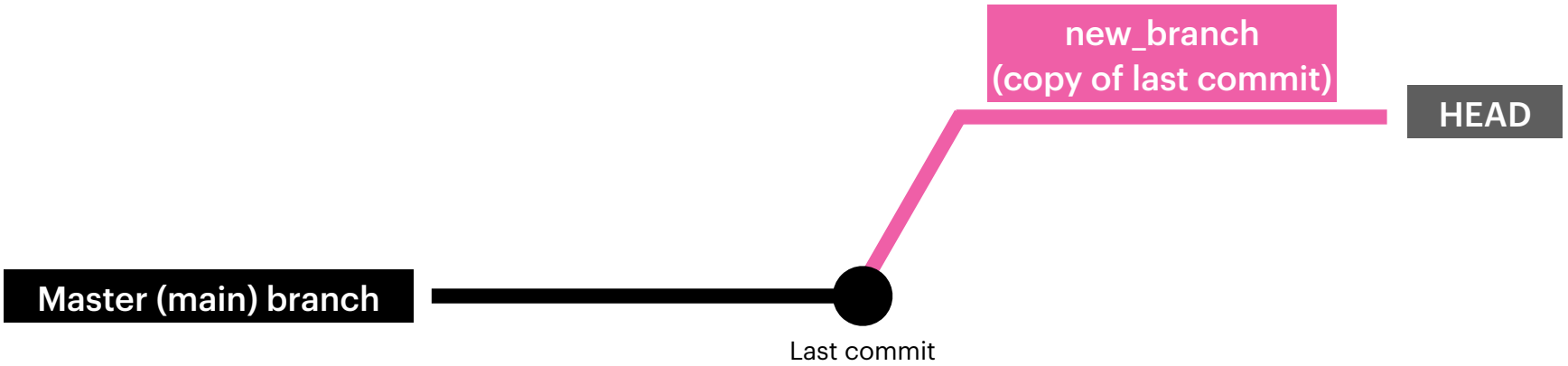


$ git branch List of branches (Master)

$ git branch new_branch

$ git checkout new_branch

Branching



$ git branch List of branches (Master)

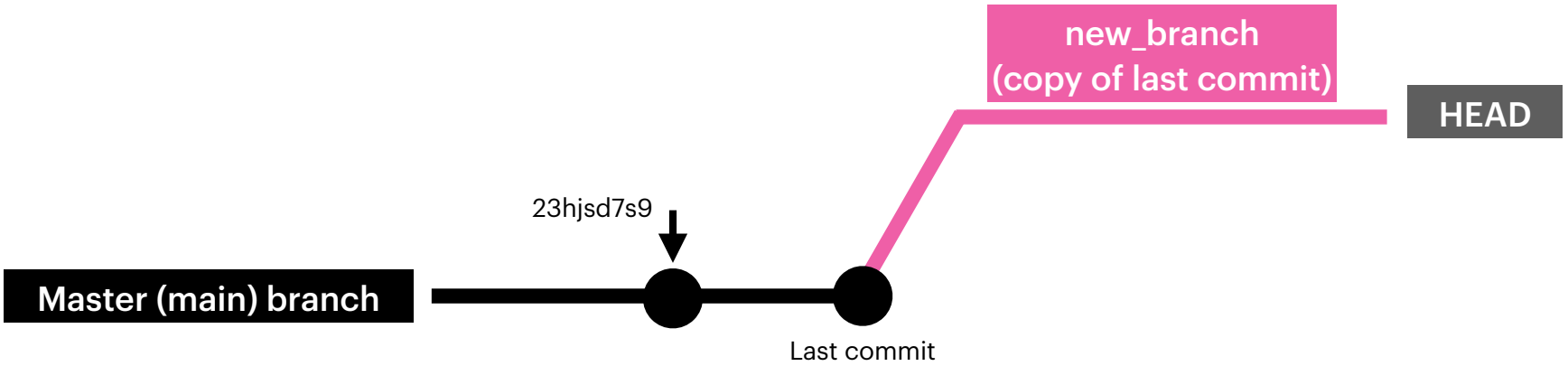
$ git branch new_branch

$ git checkout new_branch

} $ git checkout -b new_branch

$ git switch new_branch

Branching



$ git branch List of branches (Master)

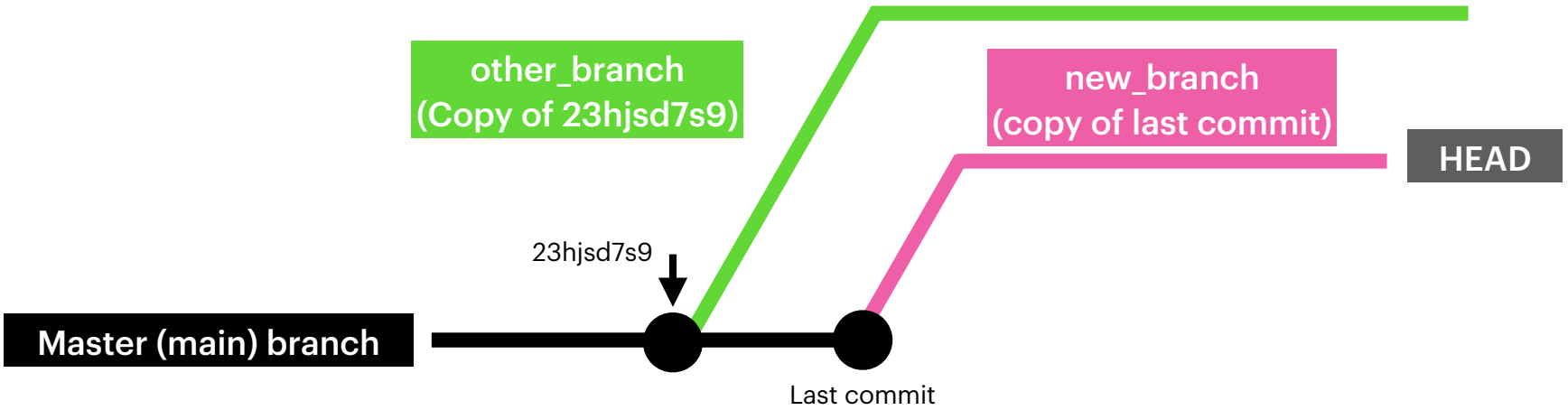
$ git branch new_branch

$ git checkout new_branch

} $ git checkout -b new_branch

$ git switch new_branch

Branching



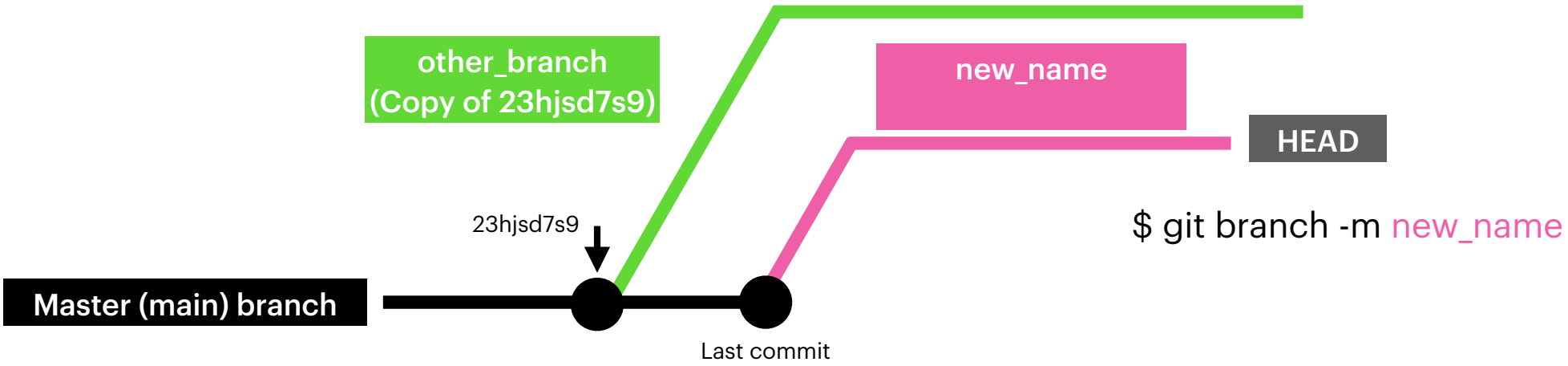
$ git branch List of branches (Master)

$ git branch new_branch
$ git checkout new_branch
} $ git checkout -b new_branch

$ git switch new_branch

$ git branch other_branch 23hjsd7s9

Branching



$ git branch List of branches (Master)

$ git branch new_branch

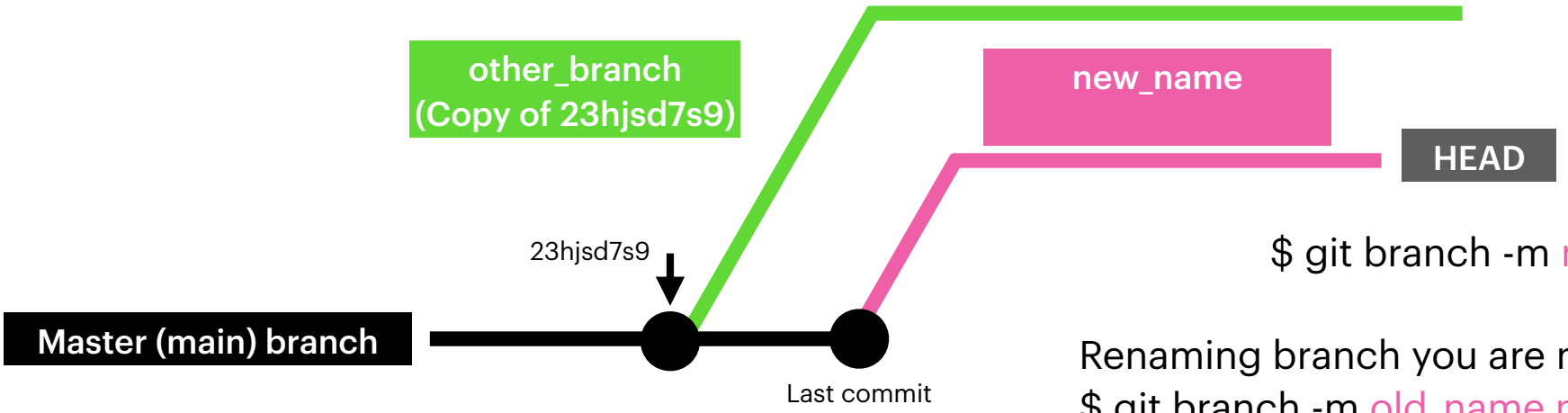
$ git checkout new_branch

} $ git checkout -b new_branch

$ git switch new_branch

$ git branch other_branch 23hjsd7s9

Branching



$ git branch -m new_name

Renaming branch you are not on:
$ git branch -m old_name new_branch_name

$ git branch List of branches (Master)

$ git branch new_branch

$ git checkout new_branch

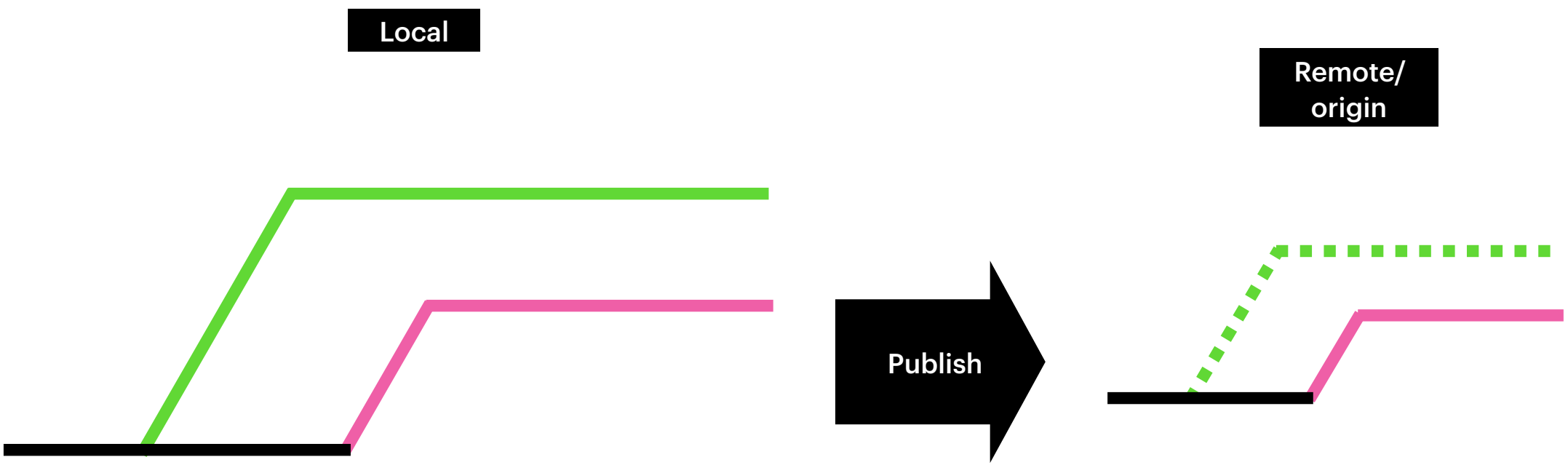
} $ git checkout -b new_branch

$ git switch new_branch

$ git branch other_branch 23hjsd7s9

Branching

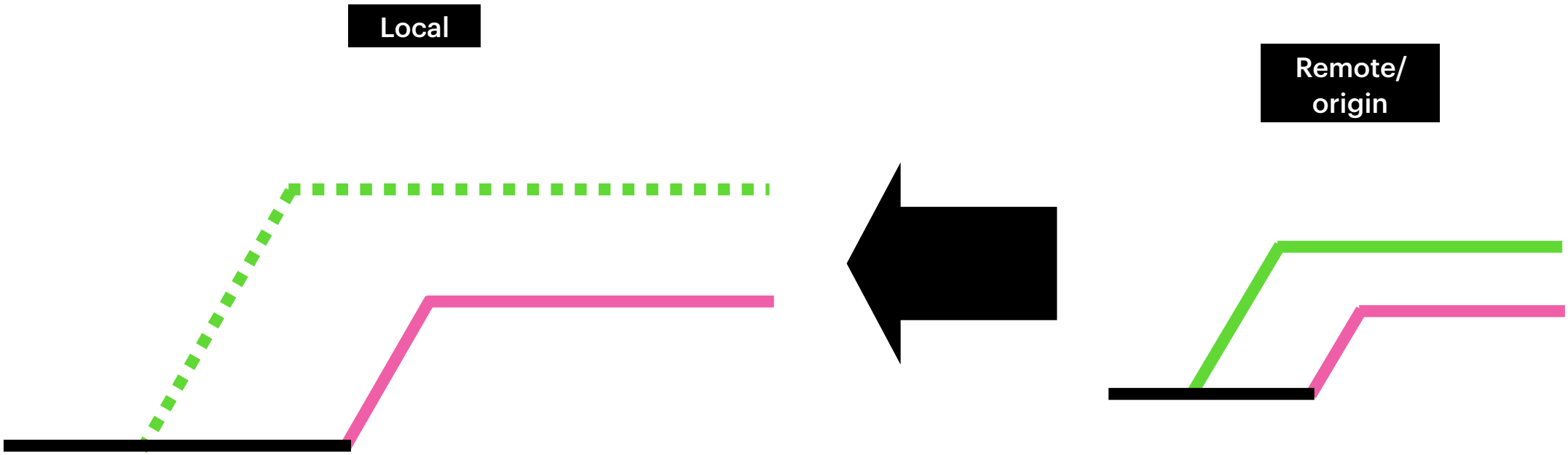
Connect local branch to remote



$ git push -u origin other_branch

Branching

Connect remote branch to local

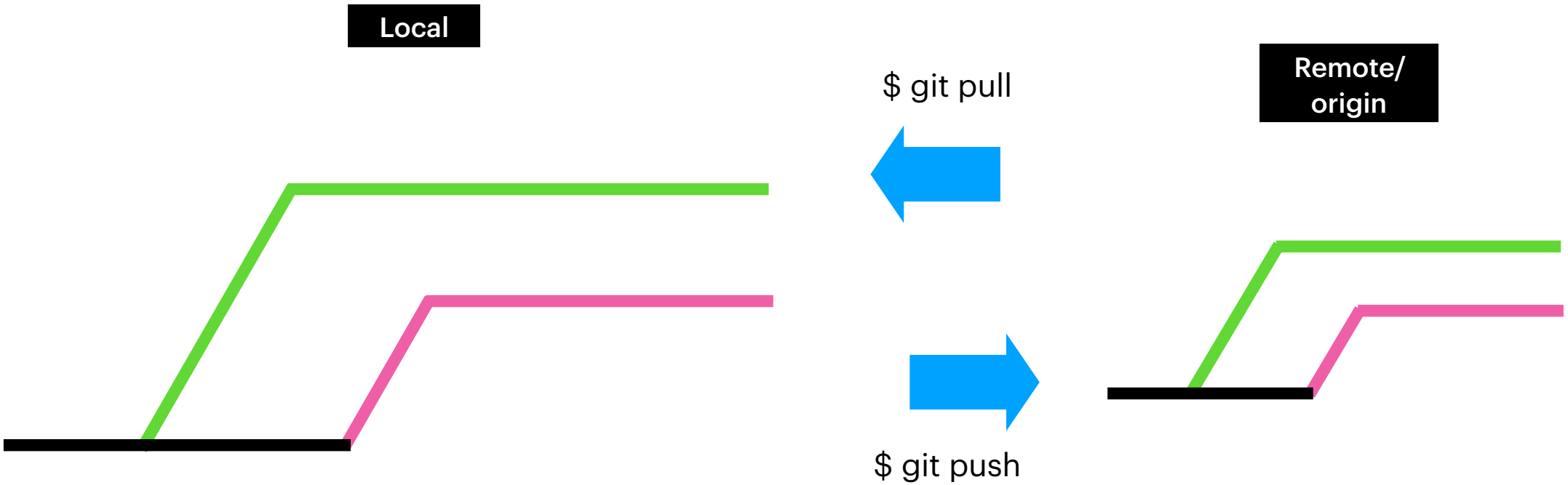


$ git branch --track new-branch origin/new-branch

$ git checkout --track origin/new-branch

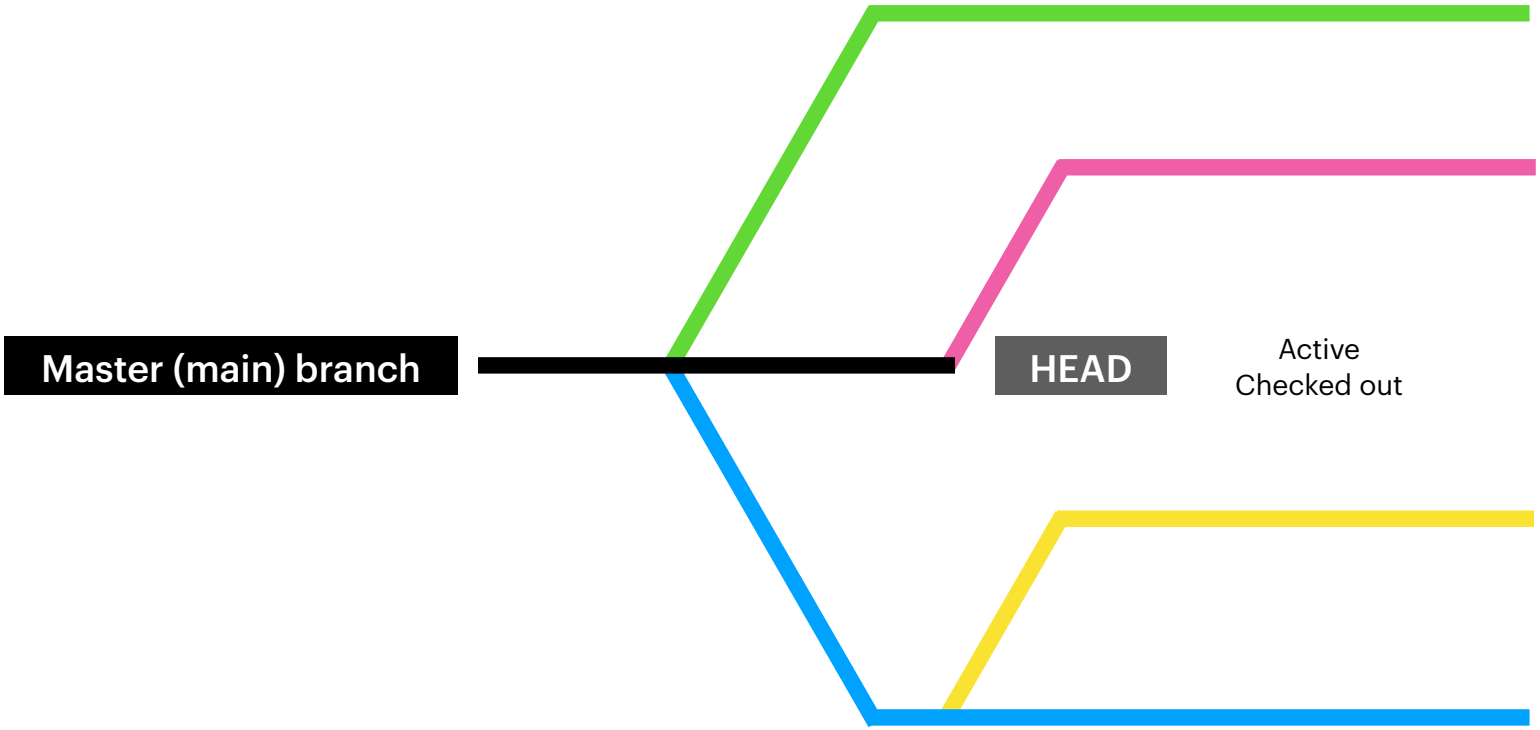
Branching

Sync local/remote branches



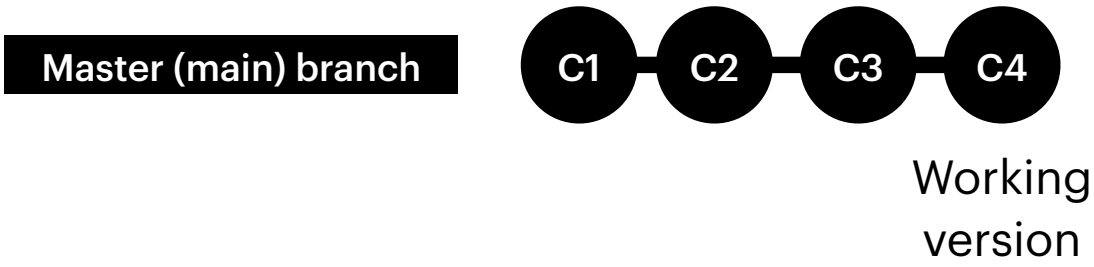
Branching

Branches can advance independently



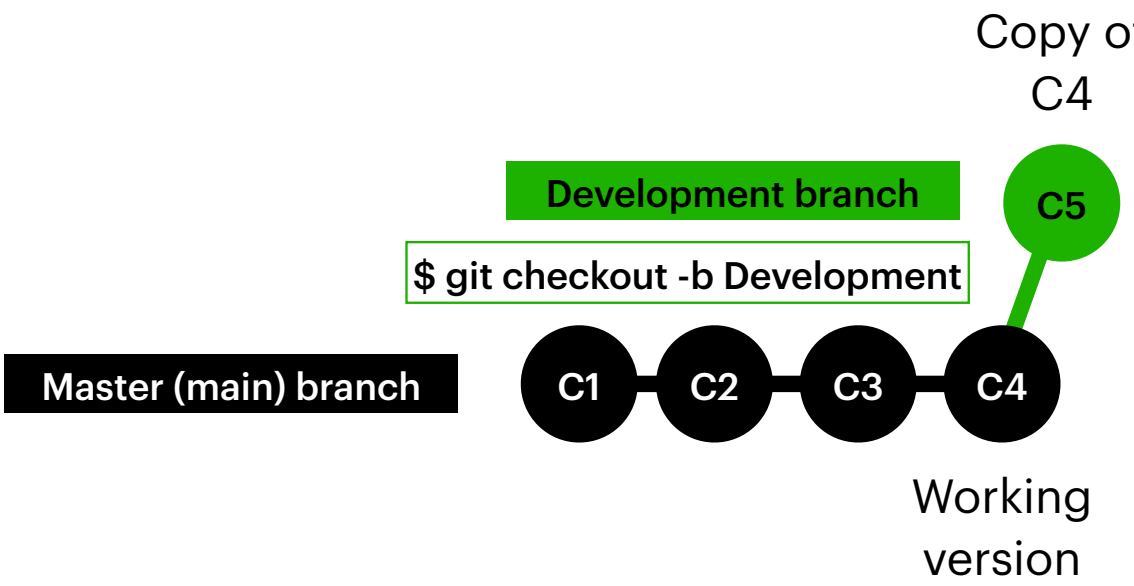
Branching

Merge



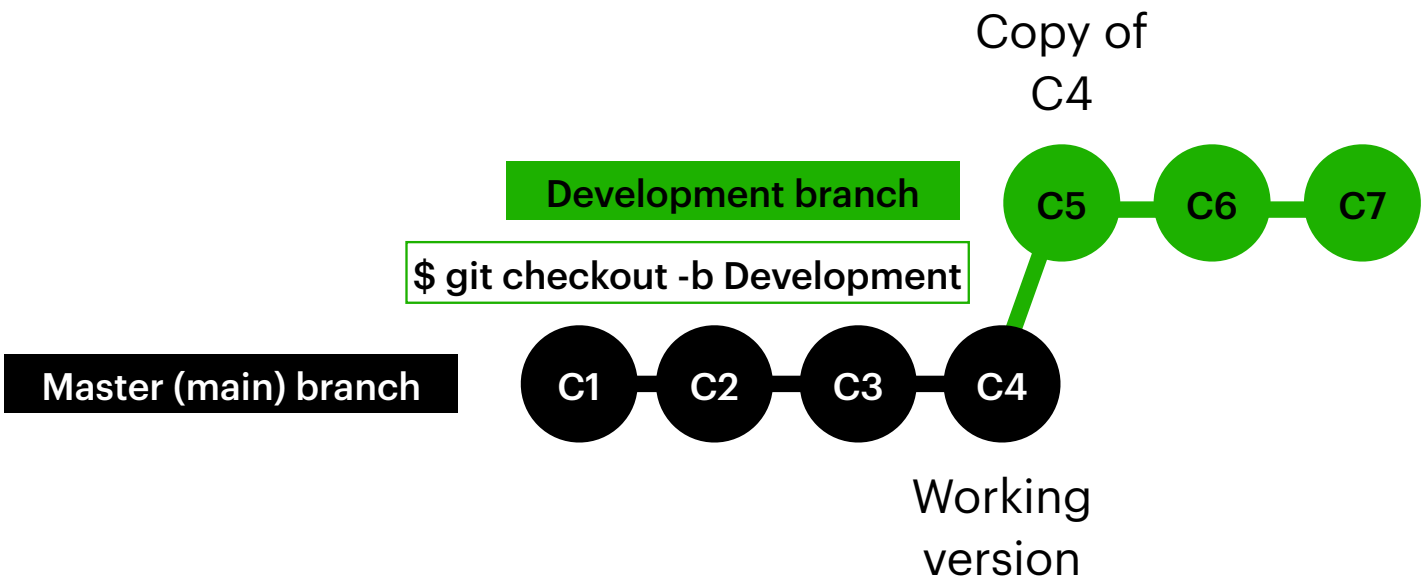
Branching

Merge



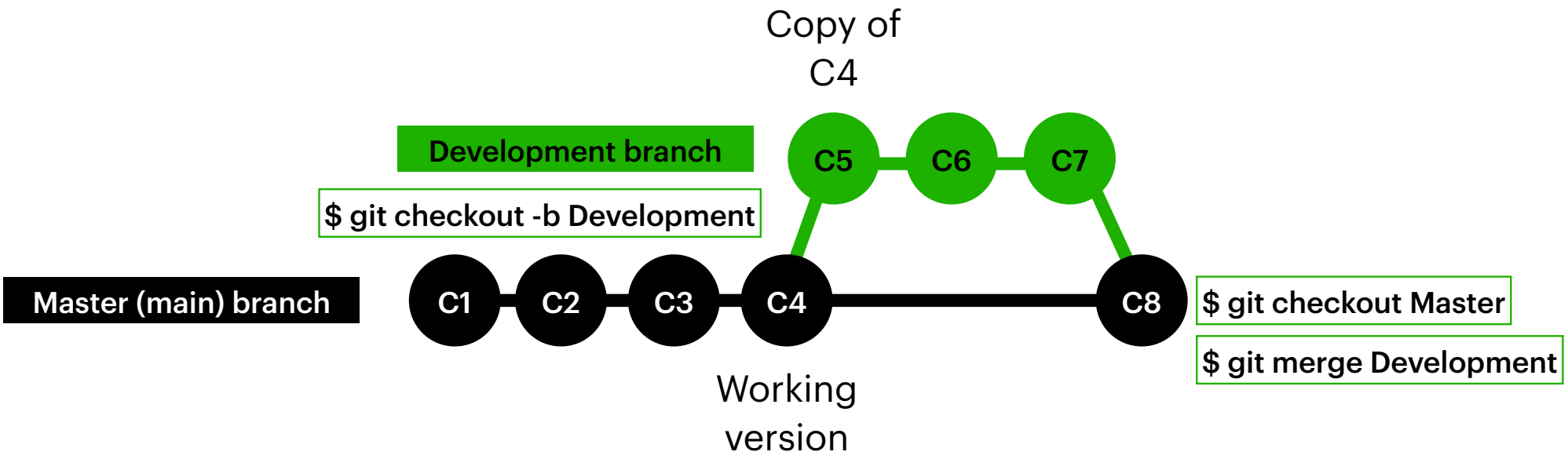
Branching

Merge



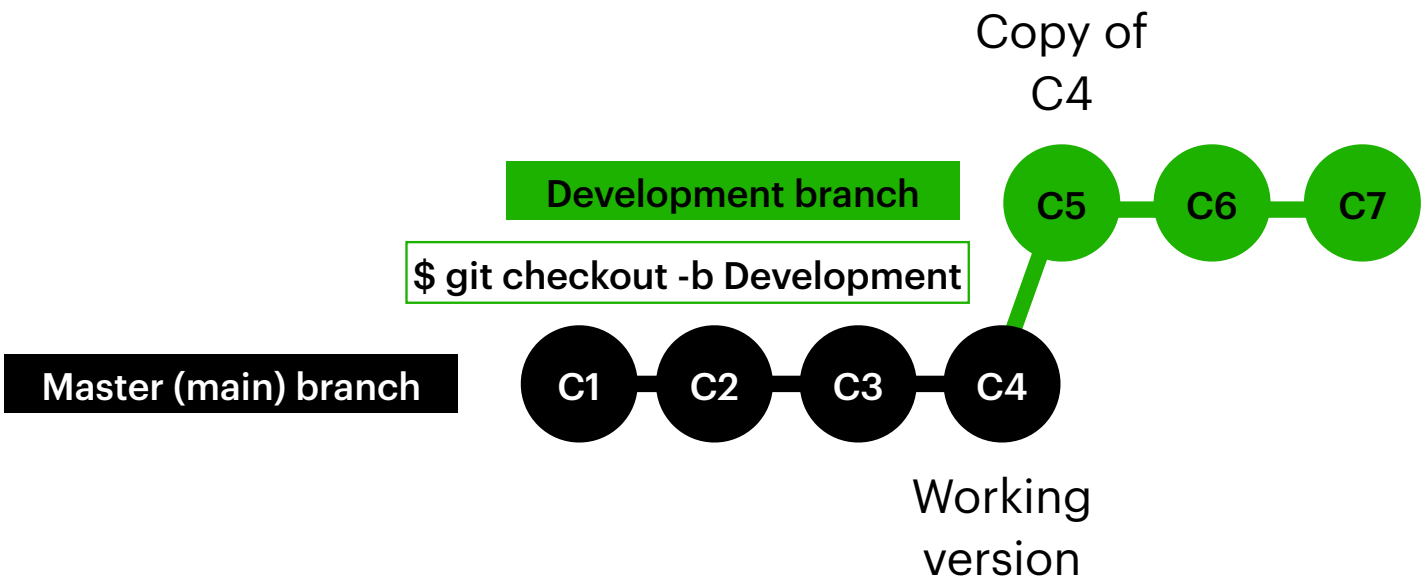
Branching

Merge



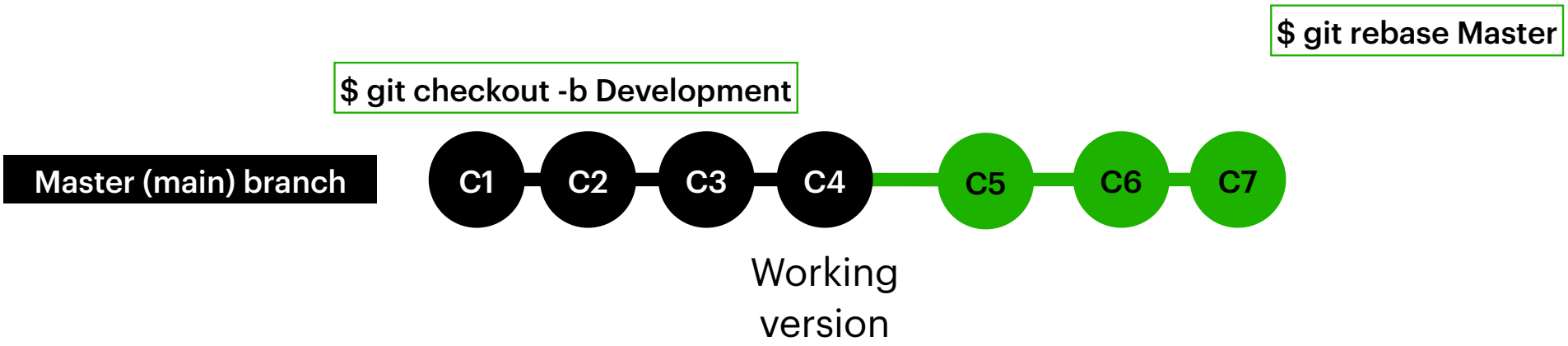
Branching

Rebase



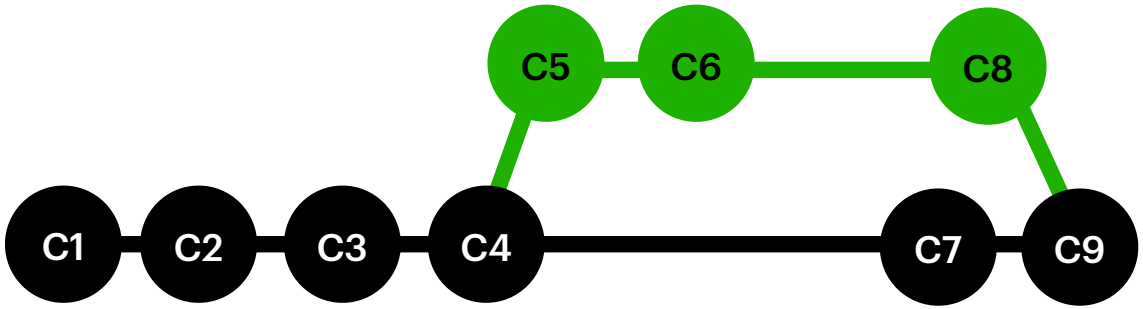
Branching

Rebase



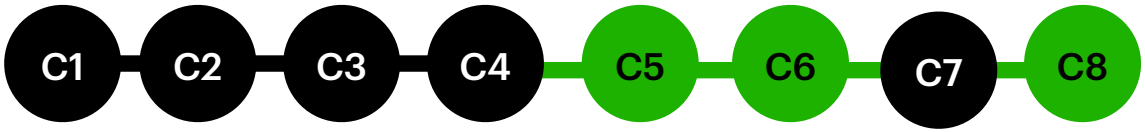
Rebase vs. Merge

- Simple and familiar
- Preserves complete history and chronological order
- Maintains the context of the branch



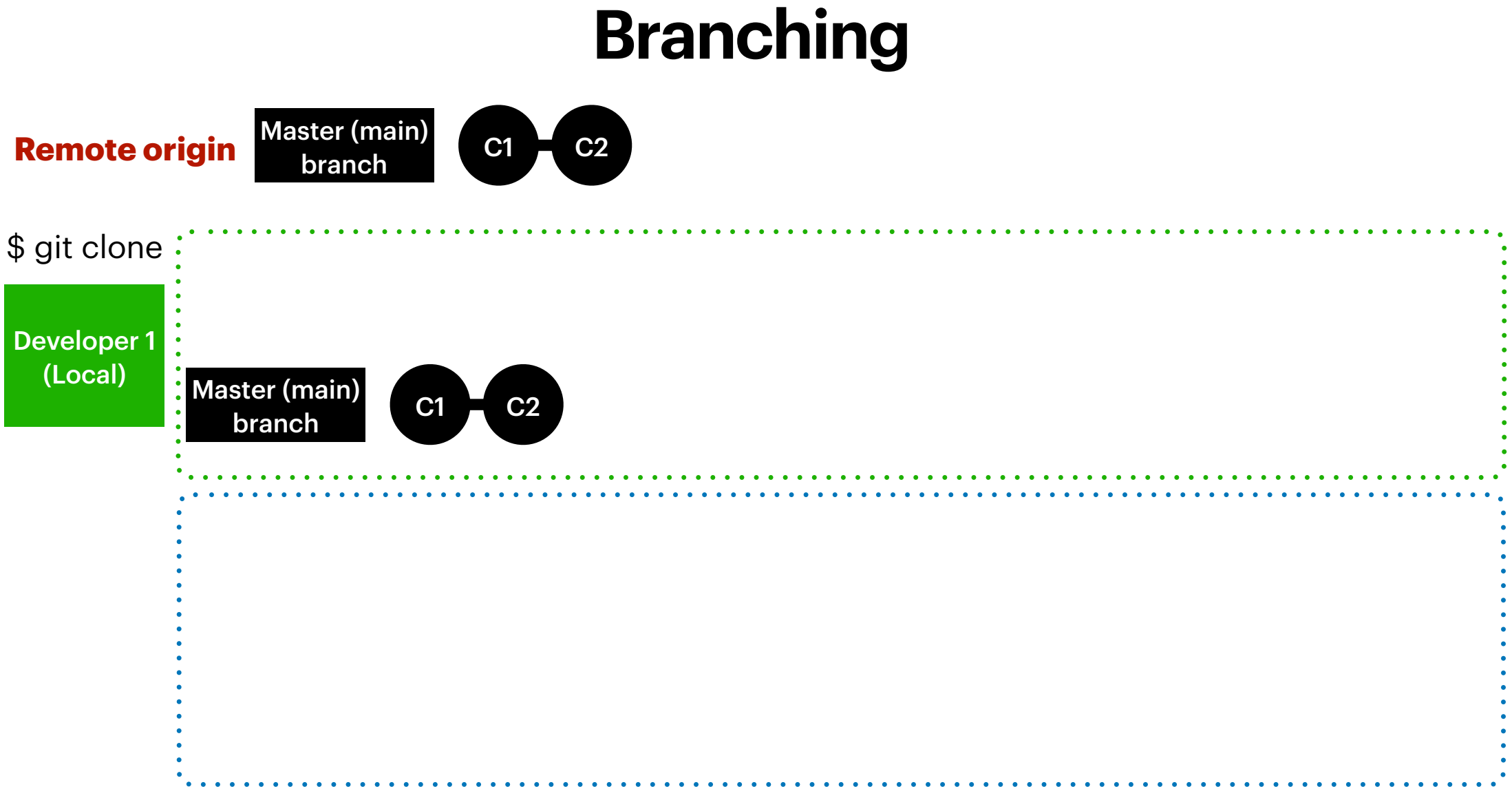
Comes down to team policy

- Streamlines a potentially complex history
- Avoids merge commit “noise” in busy repos

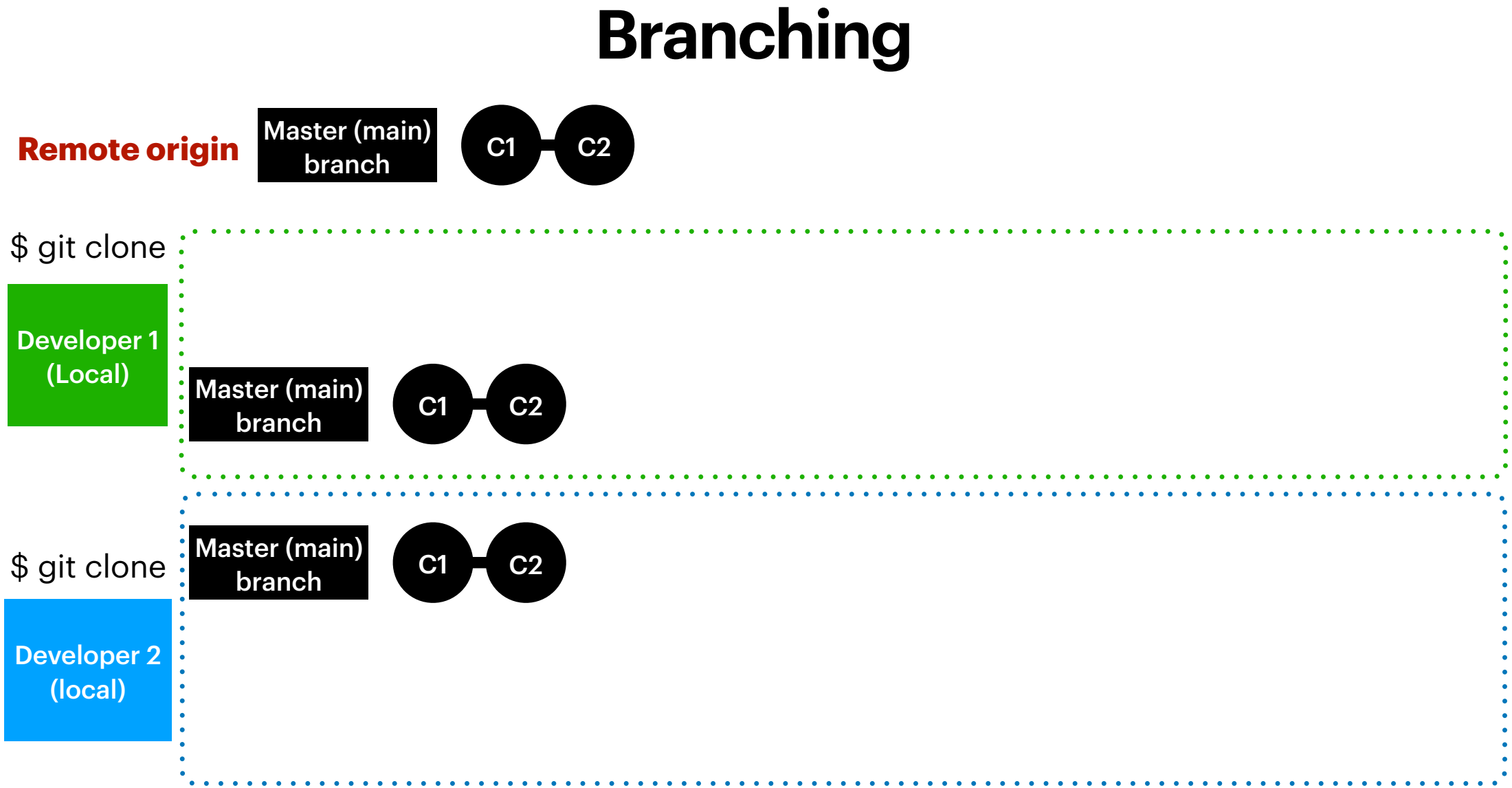


Branching

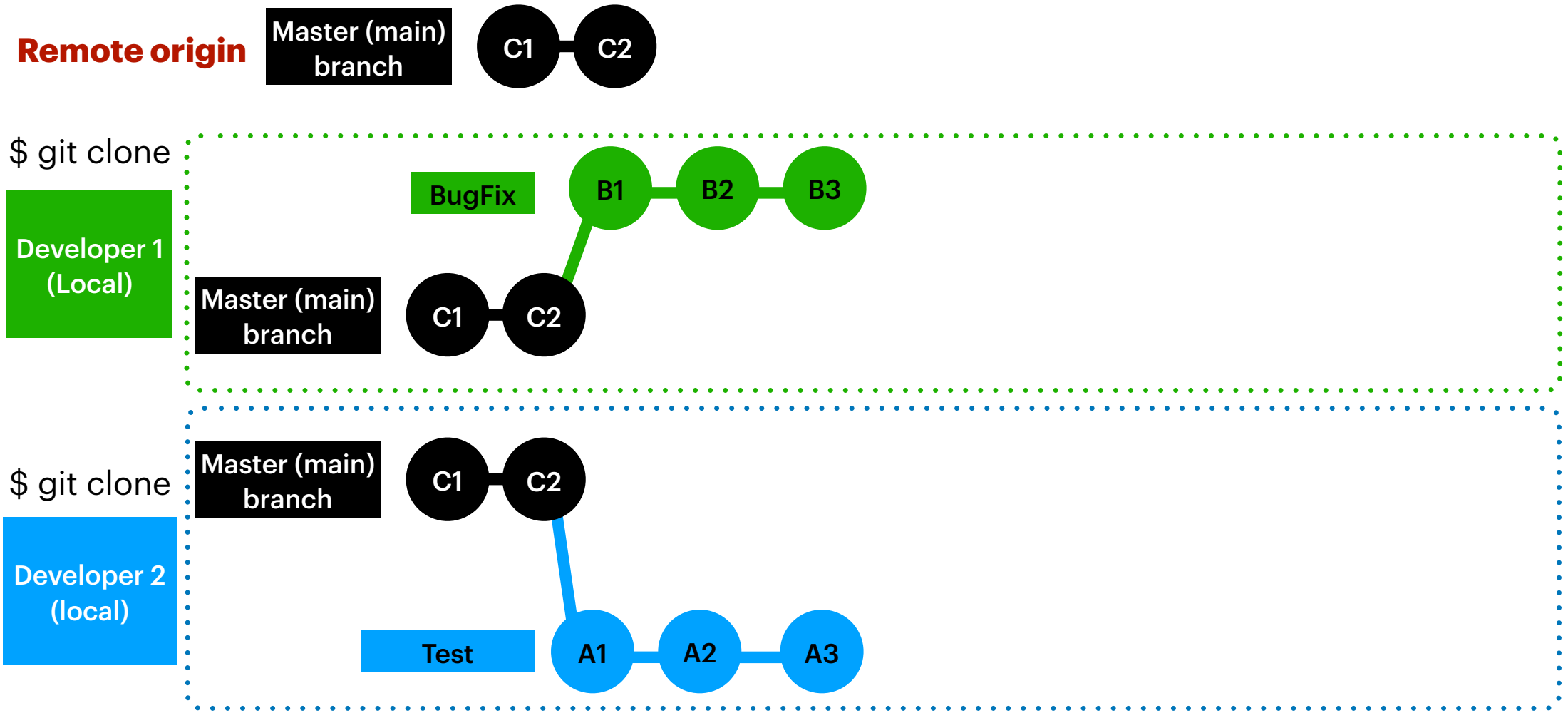
Branching


Remote origin
Master (main) branch
C1
C2
$ git clone
Developer 1 (Local)
Master (main) branch
C1
C2

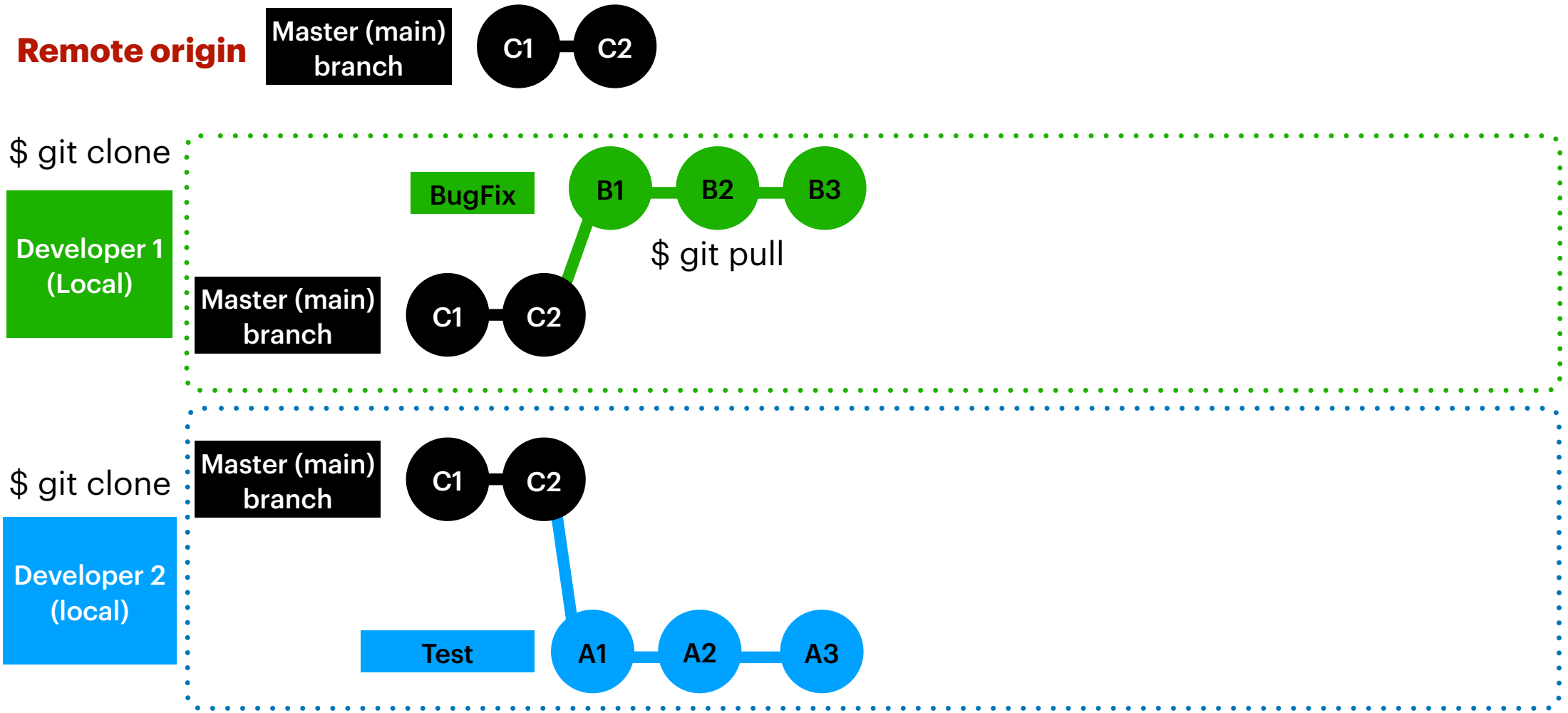
Branching


Remote origin
Master (main) branch
C1
C2
$ git clone
Developer 1 (Local)
Master (main) branch
C1
C2
$ git clone
Master (main) branch
C1
C2
Developer 2 (local)

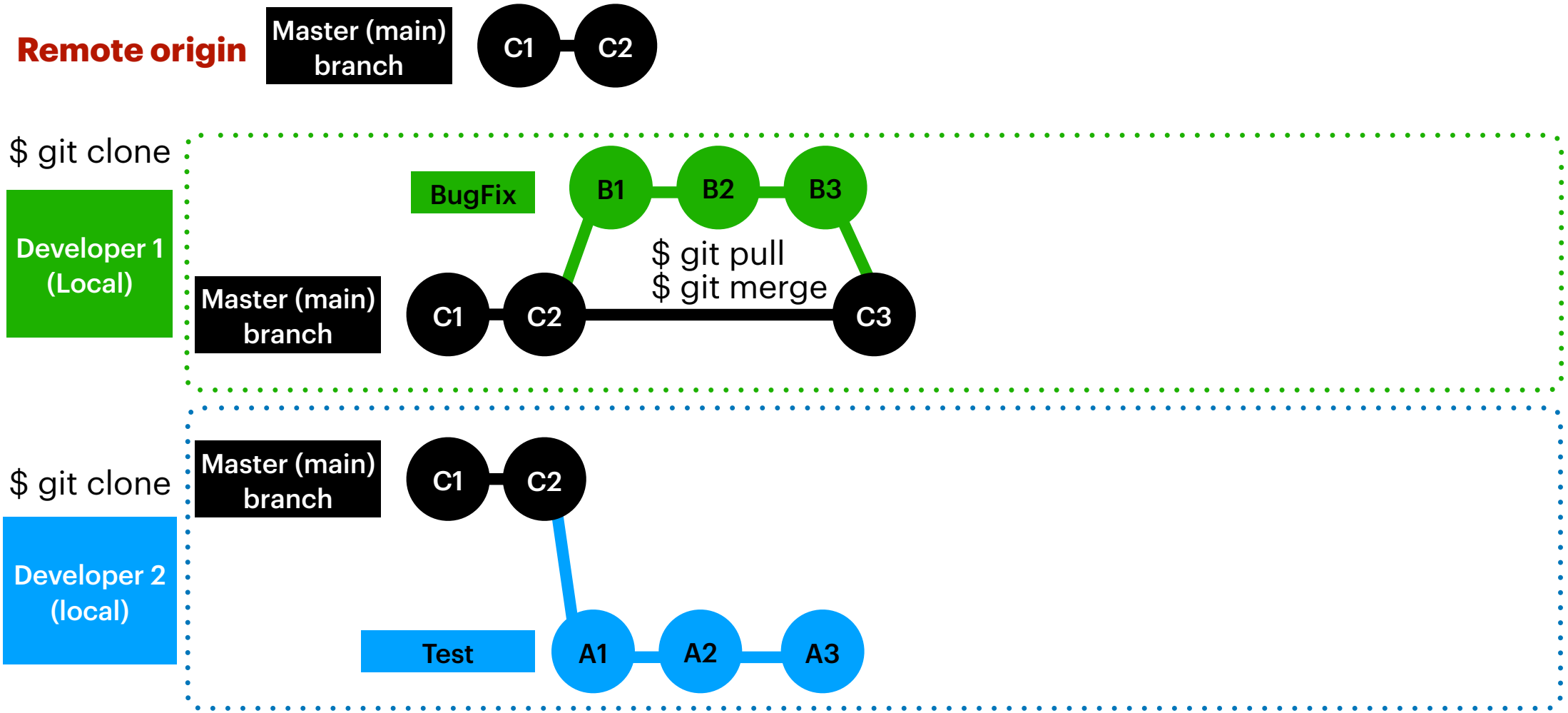
Branching


Remote origin
Master (main) branch
C1
C2
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
Master (main) branch
C1
C2
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
Test
A1
A2
A3

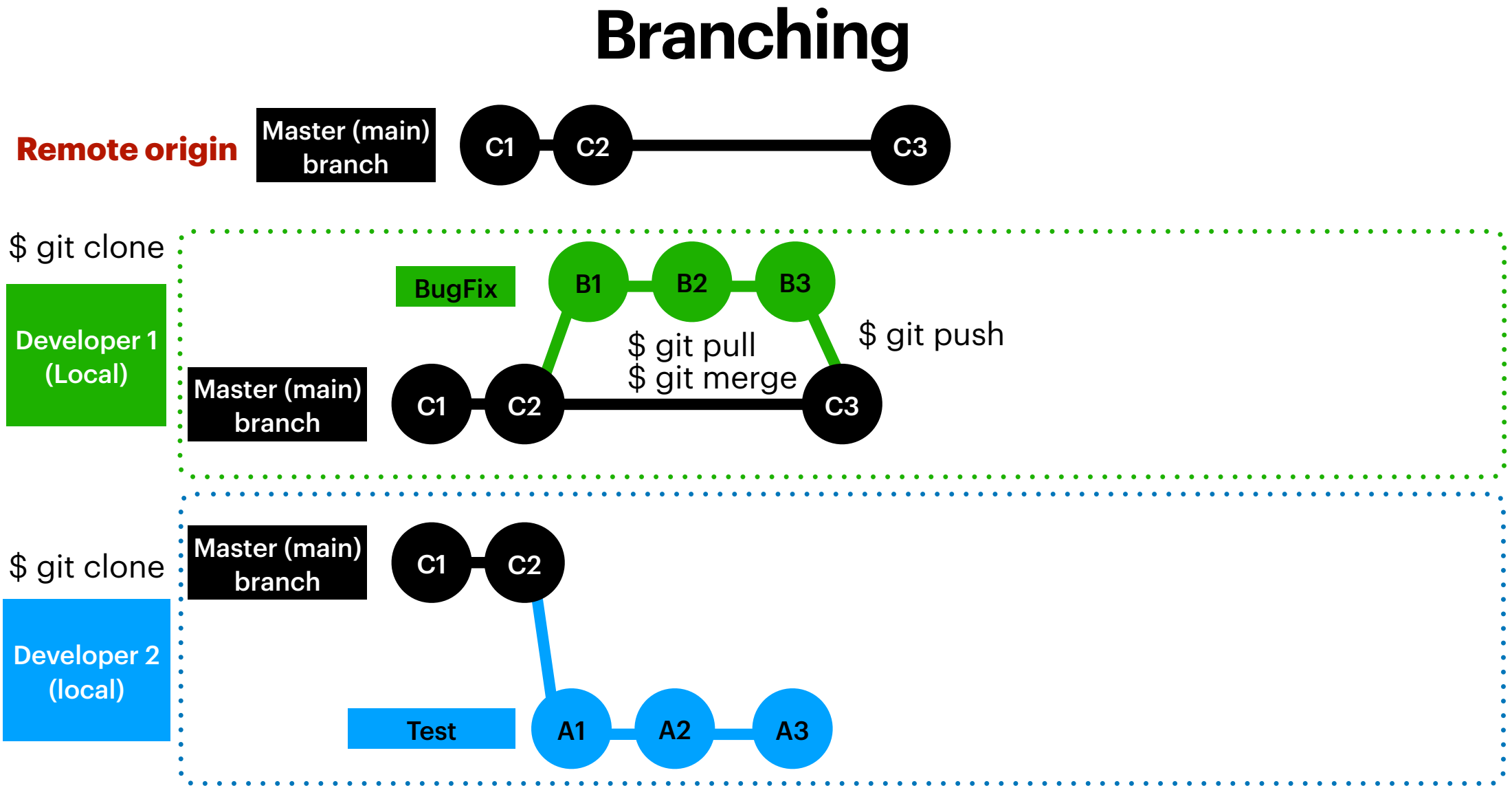
Branching


Remote origin
Master (main) branch
C1
C2
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
Master (main) branch
C1
C2
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
Test
A1
A2
A3

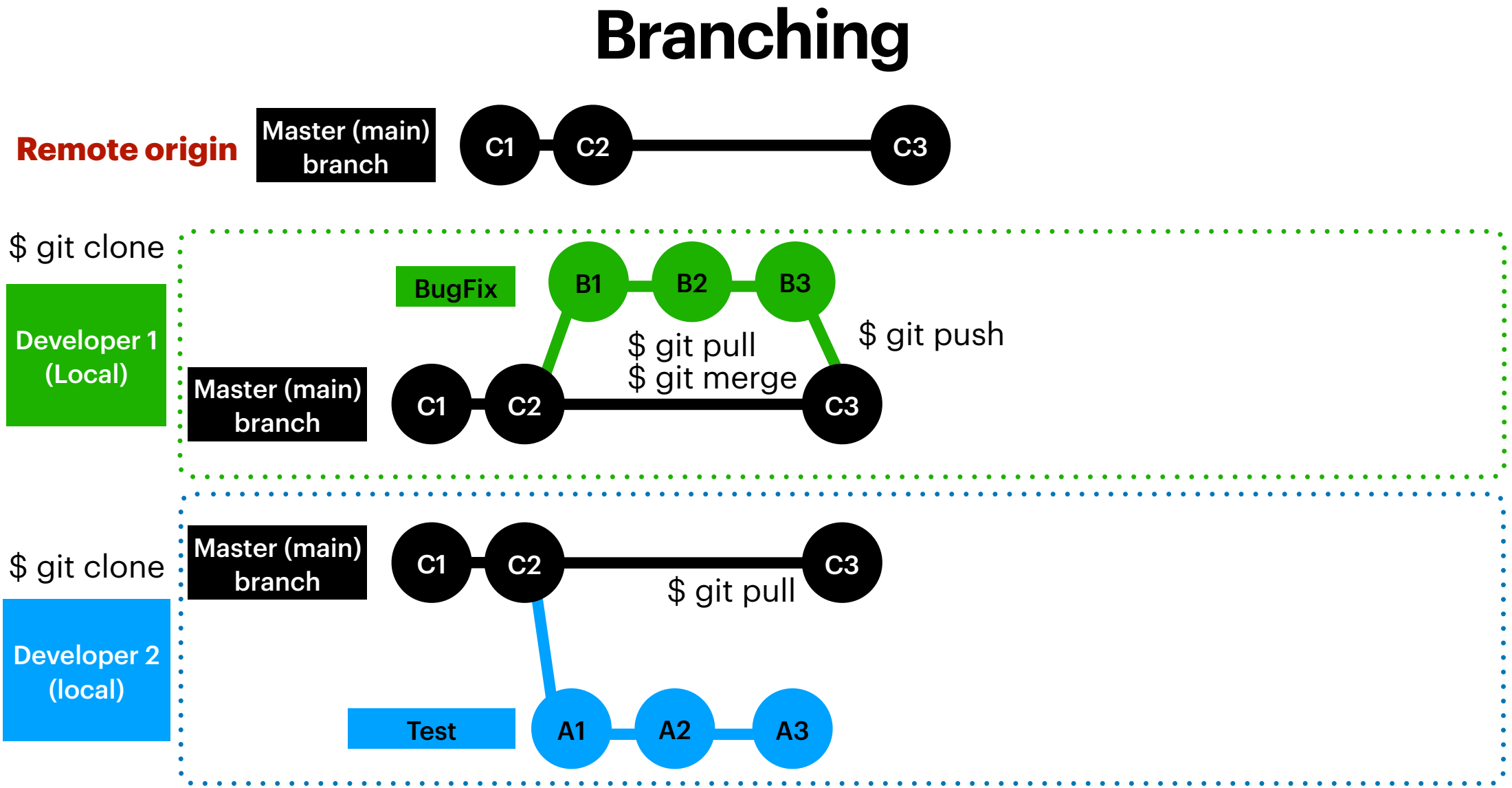
Branching


Remote origin
Master (main) branch
C1
C2
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
Test
A1
A2
A3

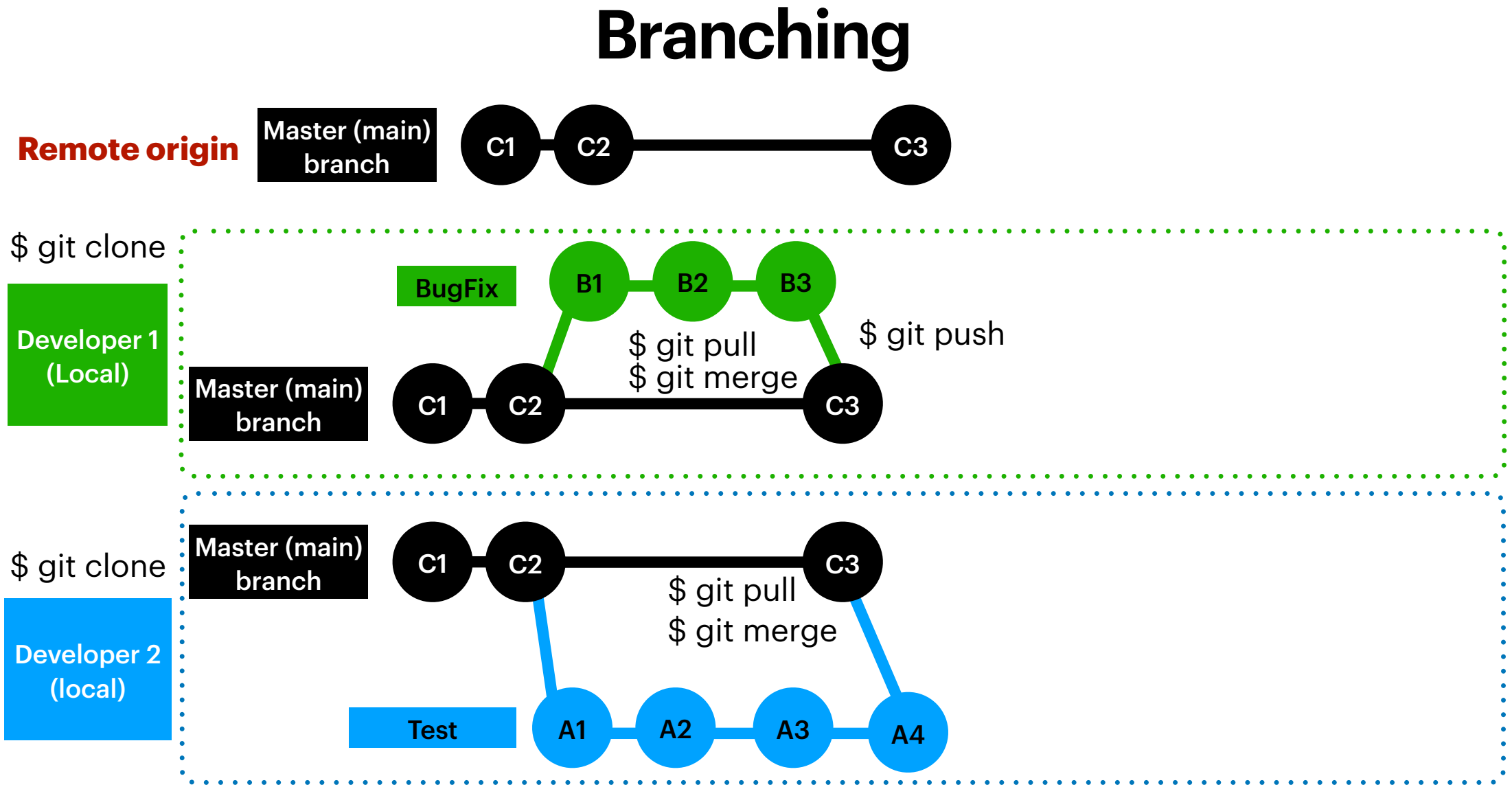
Branching


Remote origin
Master (main) branch
C1
C2
C3
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
Test
A1
A2
A3

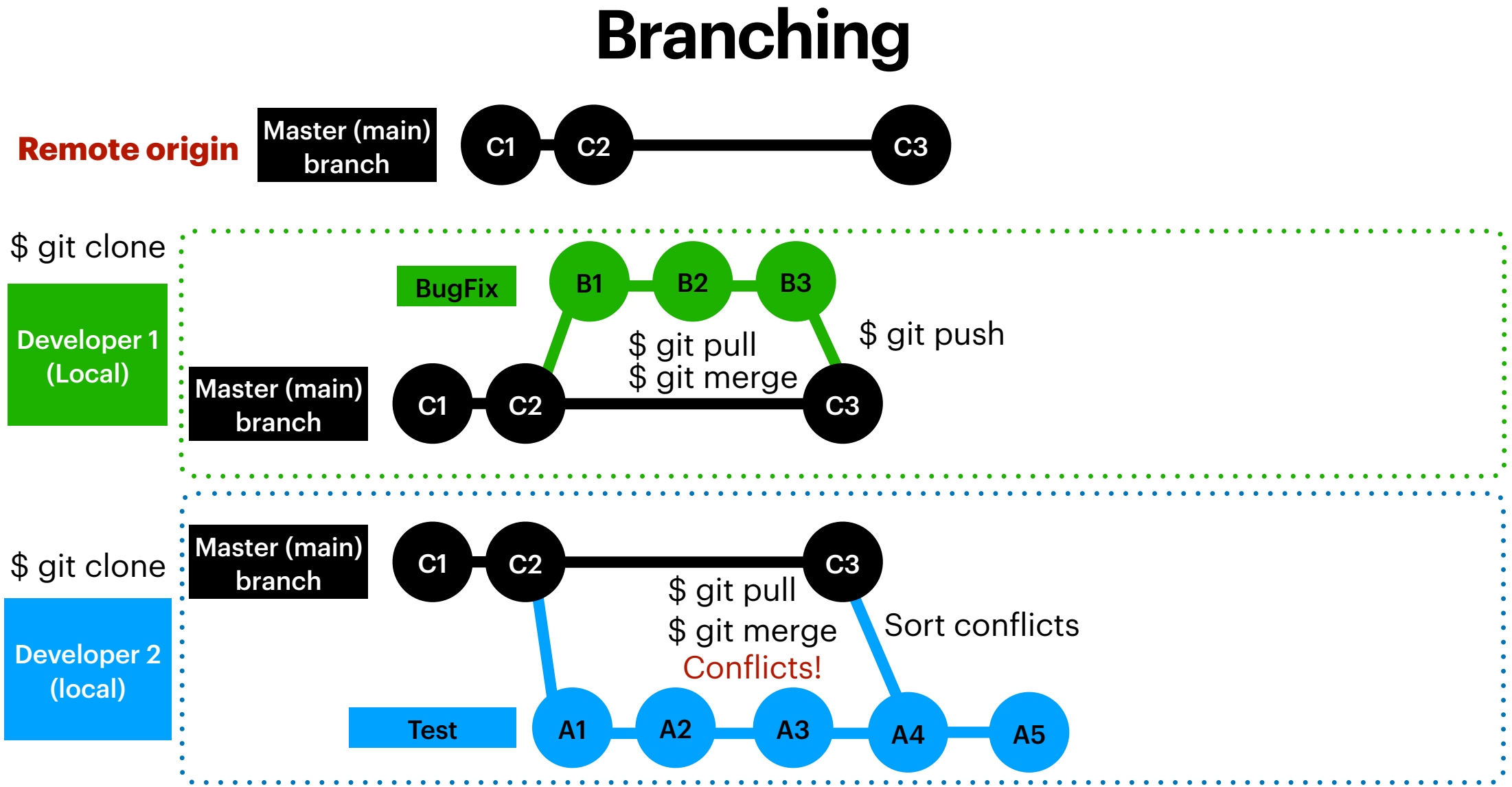
Branching


Remote origin
Master (main) branch
C1
C2
C3
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
$ git pull
Test
A1
A2
A3

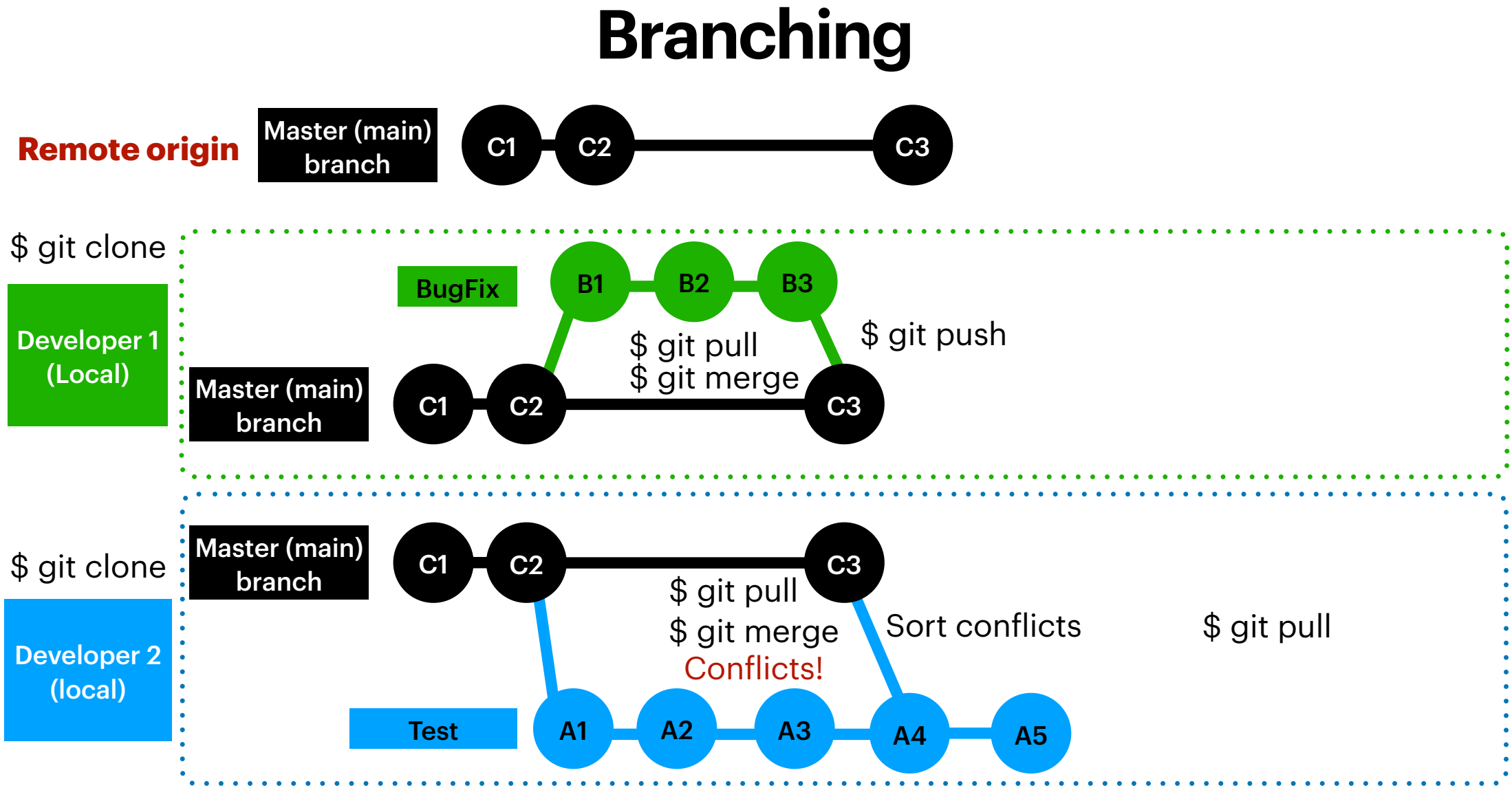
Branching


Remote origin
Master (main) branch
C1
C2
C3
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
$ git pull
$ git merge
Test
A1
A2
A3
A4

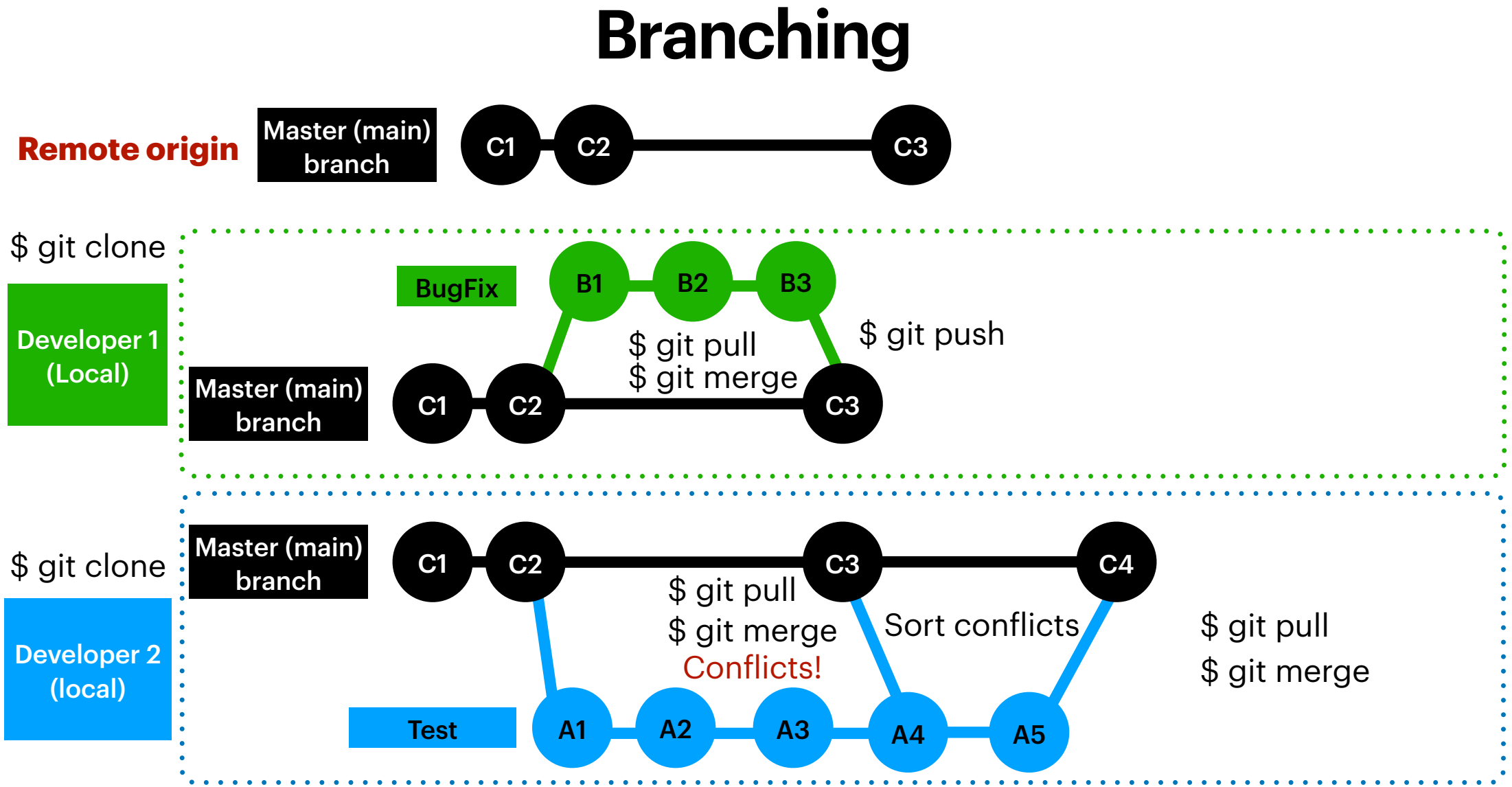
Branching


Remote origin
Master (main) branch
C1
C2
C3
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
$ git pull
$ git merge
Conflicts!
Sort conflicts
Test
A1
A2
A3
A4
A5

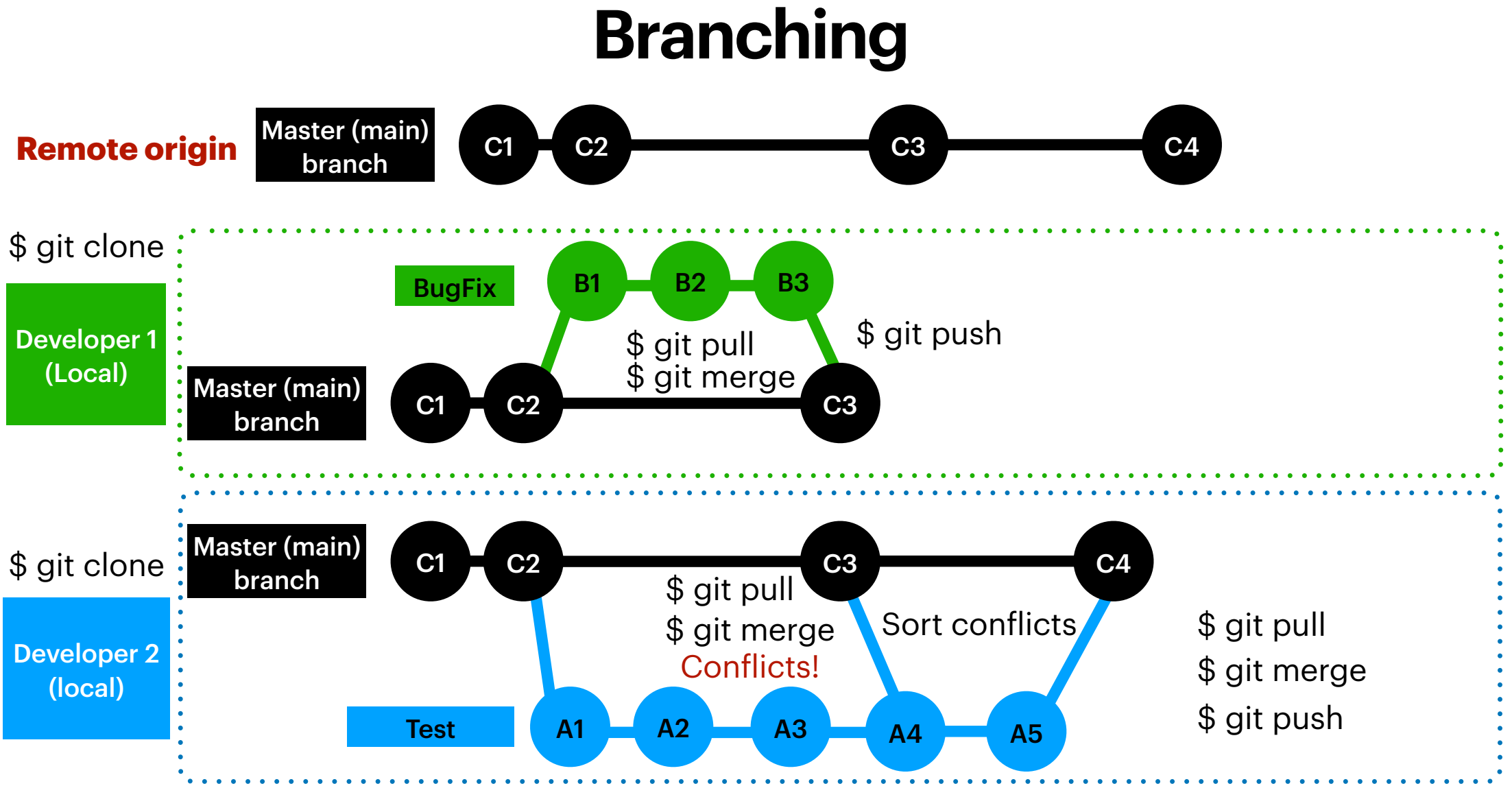
Branching


Remote origin
Master (main) branch
C1
C2
C3
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
$ git pull
$ git merge
Conflicts!
Sort conflicts
$ git pull
Test
A1
A2
A3
A4
A5

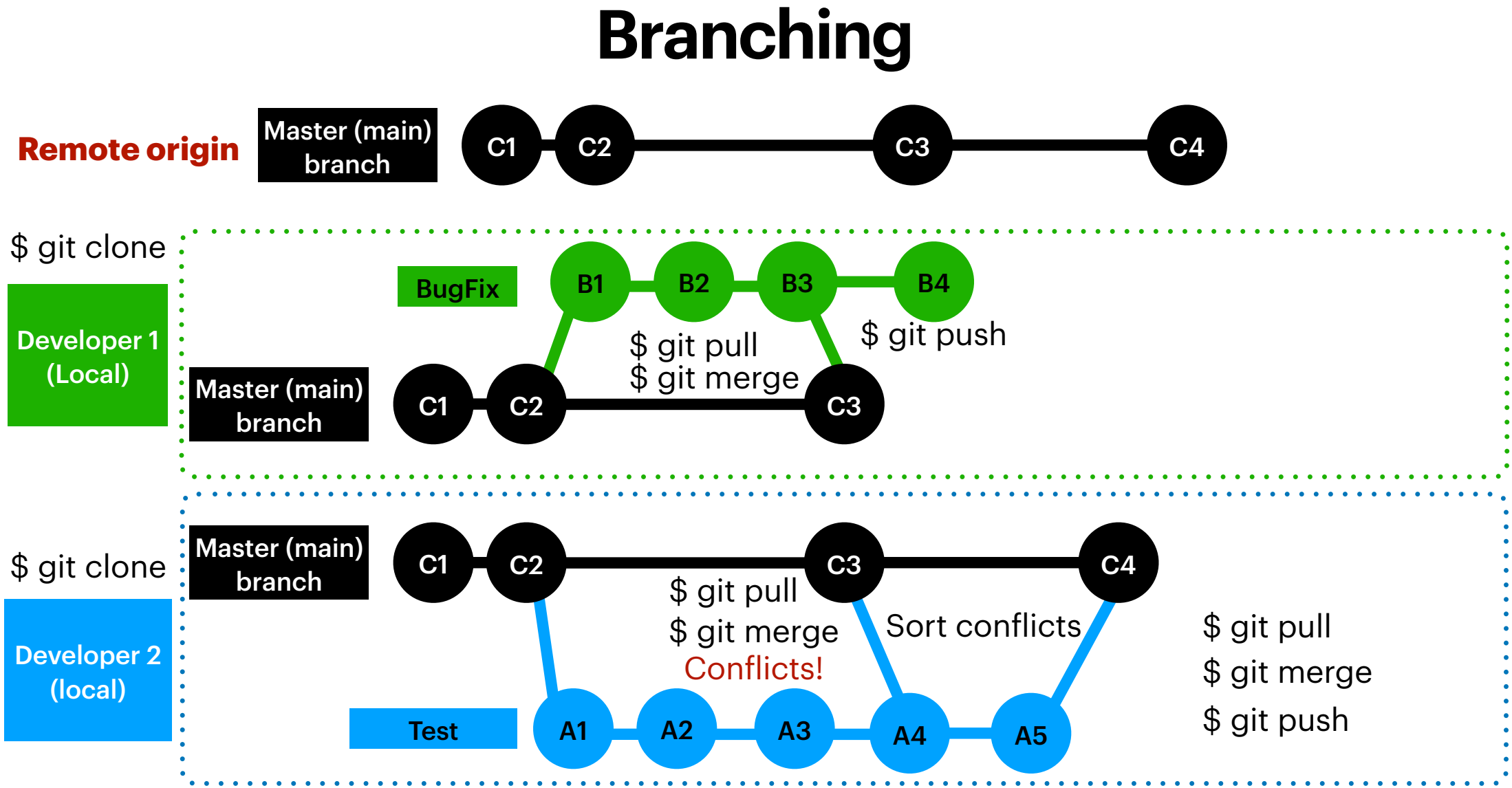
Branching


Remote origin
Master (main) branch
C1
C2
C3
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
$ git pull
$ git merge
Test
A1
A2
A3
A4
A5

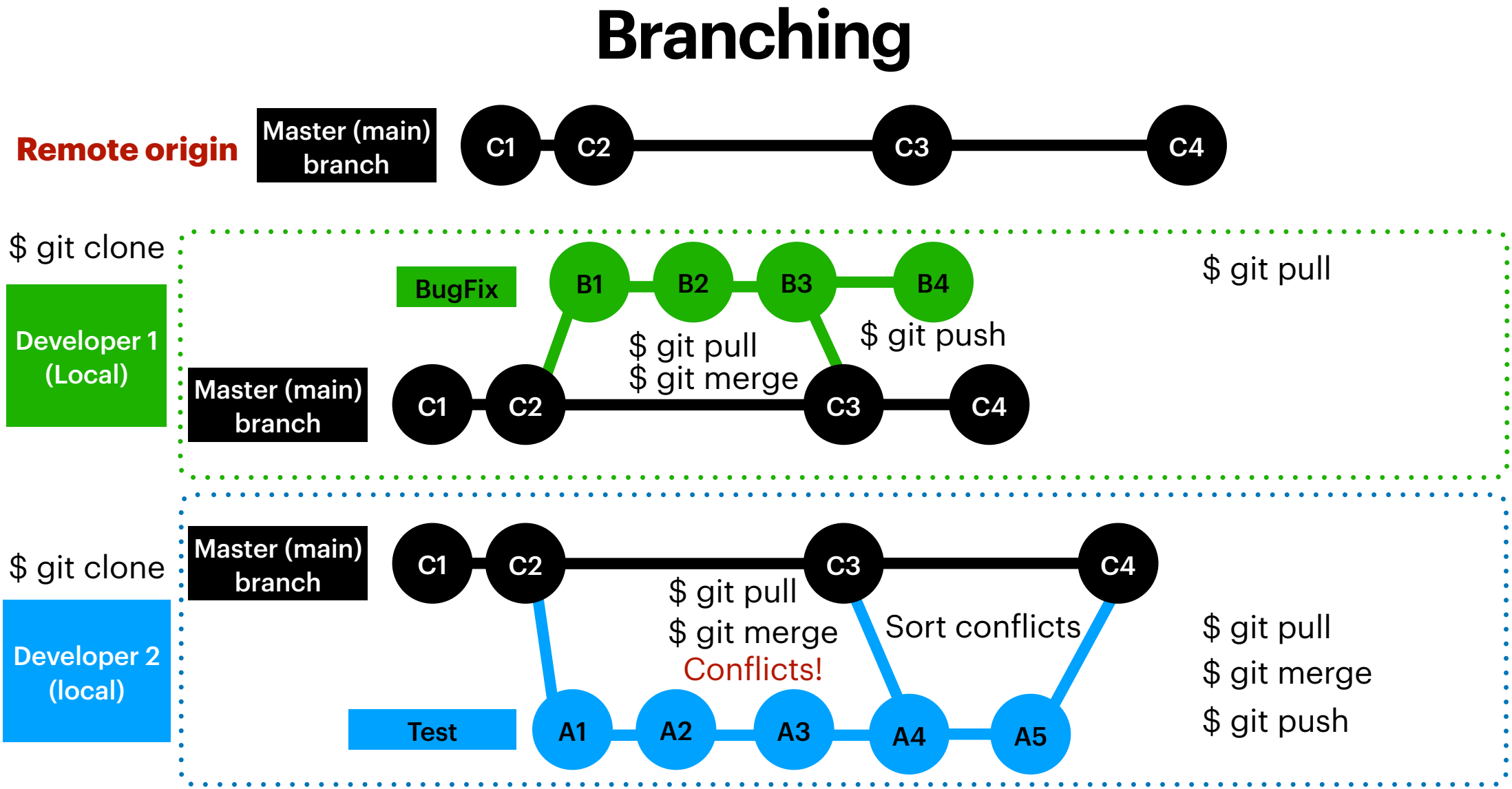
Branching


Remote origin
Master (main) branch
C1
C2
C3
C4
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
$ git pull
$ git merge
$ git push
Test
A1
A2
A3
A4
A5

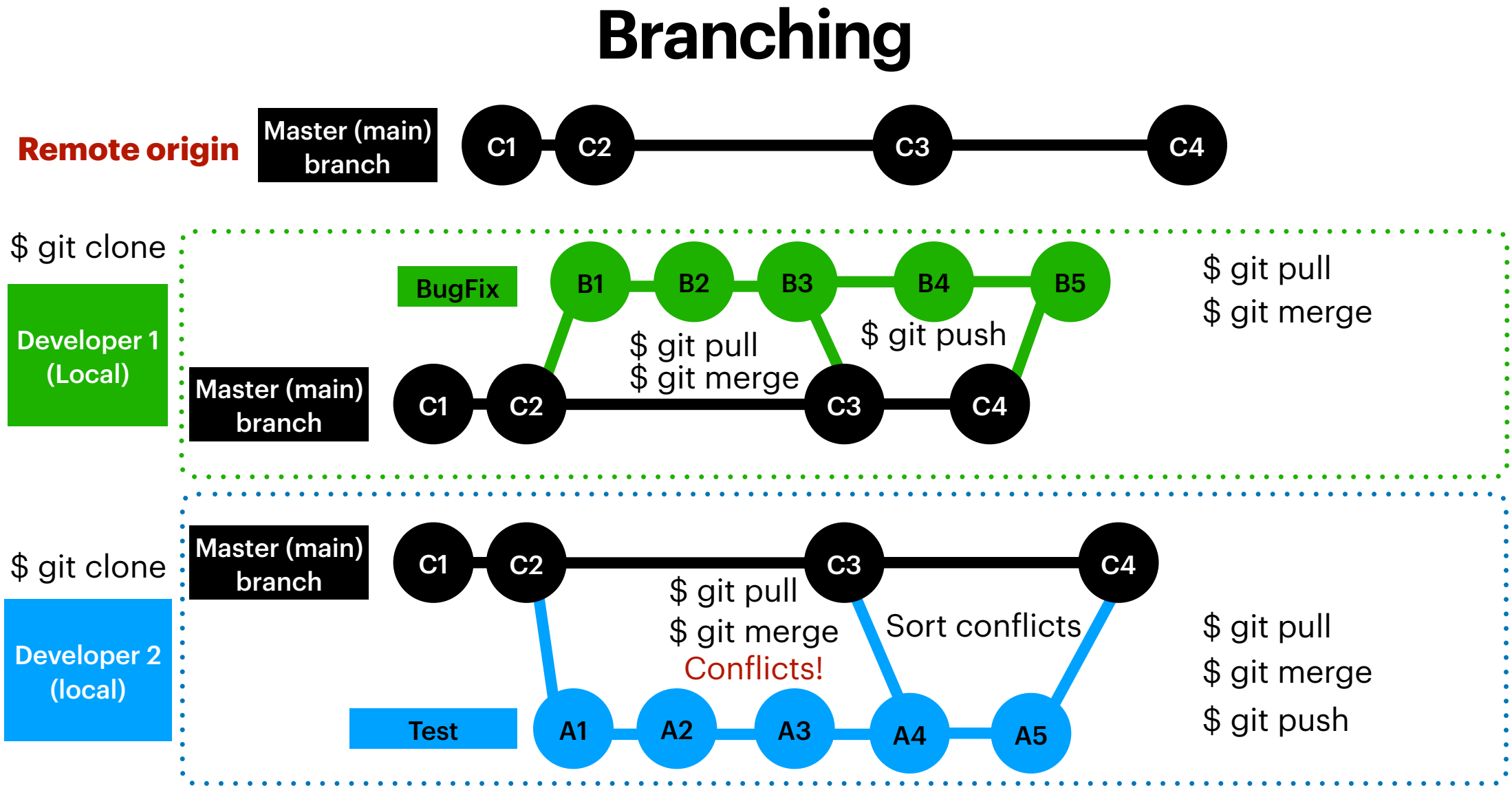
Branching


Remote origin
Master (main) branch
C1
C2
C3
C4
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
B4
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
$ git pull
$ git merge
$ git push
Test
A1
A2
A3
A4
A5

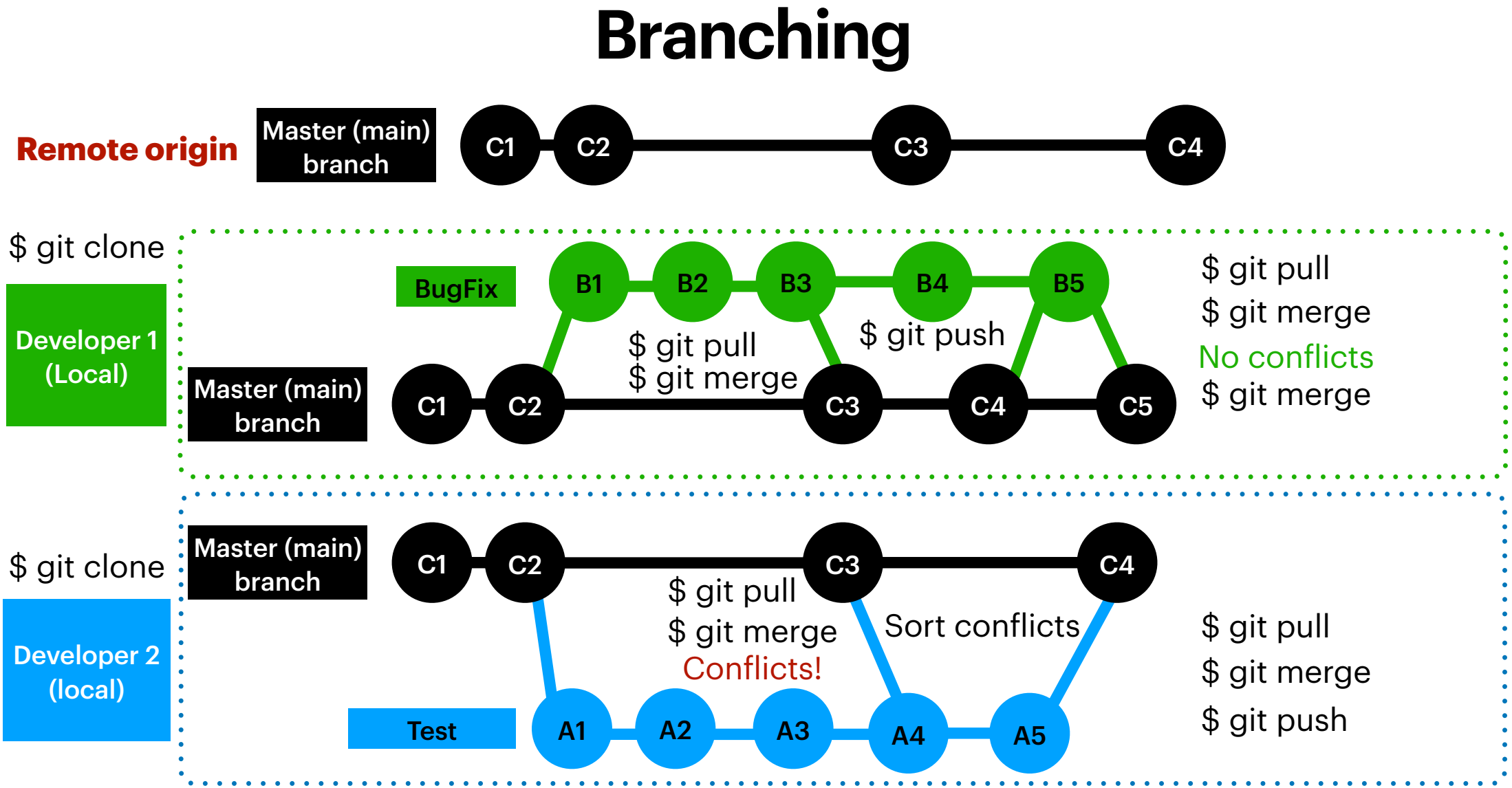
Branching


Remote origin
Master (main) branch
C1
C2
C3
C4
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
B4
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
$ git pull
$ git merge
$ git push
Test
A1
A2
A3
A4
A5

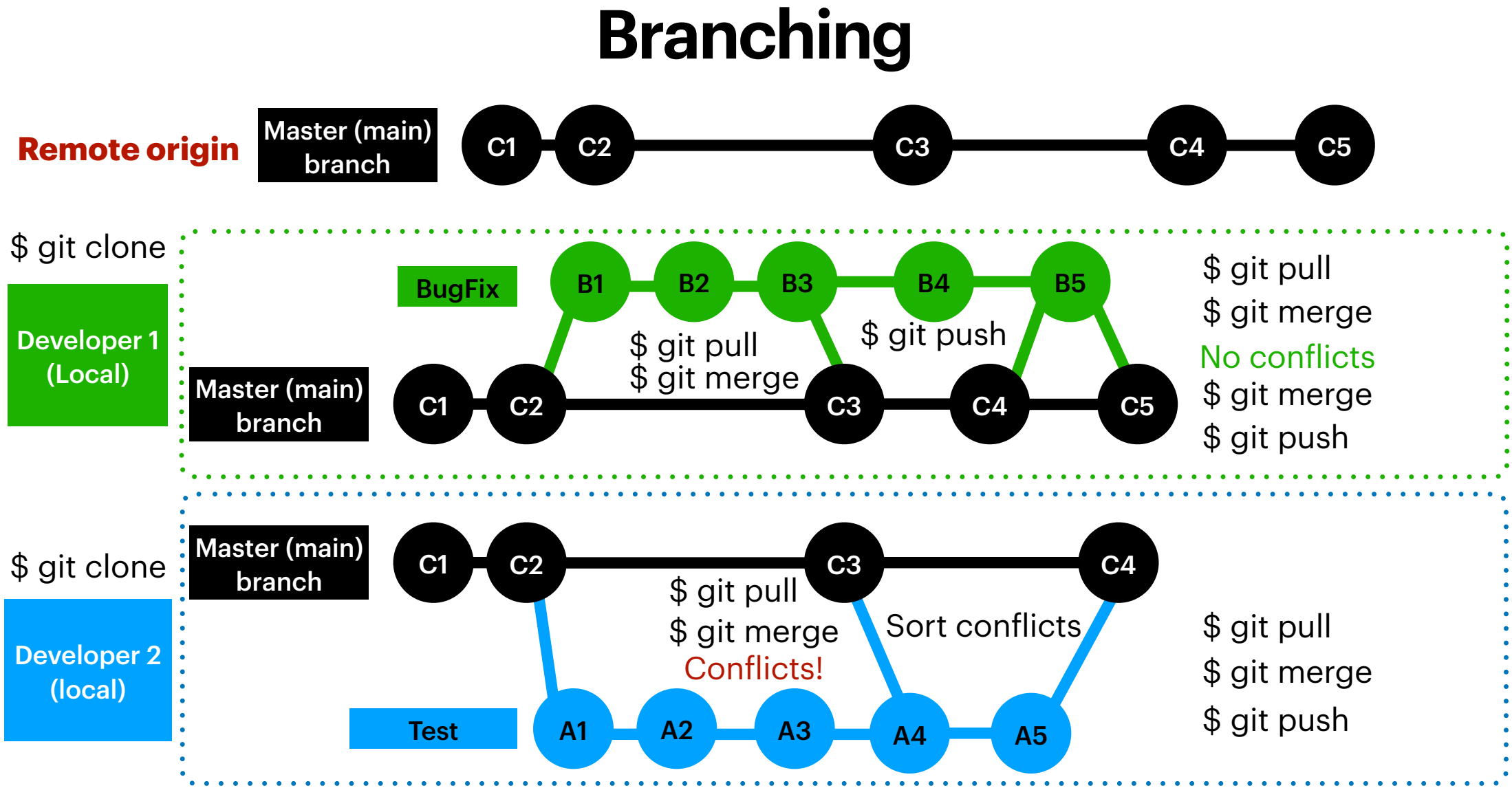
Branching


Remote origin
Master (main) branch
C1
C2
C3
C4
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
B4
B5
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
$ git pull
$ git merge
$ git push
Test
A1
A2
A3
A4
A5

Branching

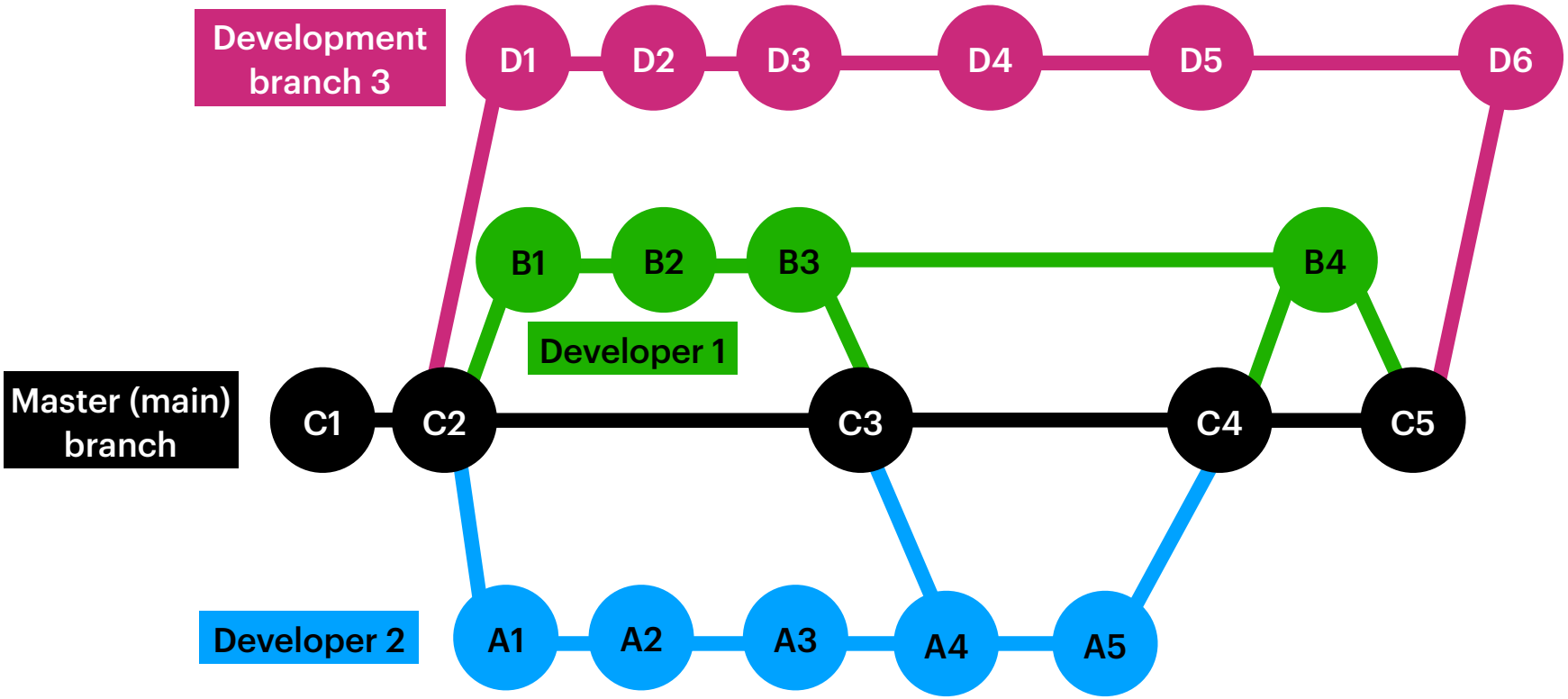

Remote origin
Master (main) branch
C1
C2
C3
C4
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
B4
B5
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
C4
C5
$ git pull
$ git merge
No conflicts
$ git merge
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
Test
A1
A2
A3
A4
A5
$ git pull
$ git merge
$ git push

Branching


Remote origin
Master (main) branch
C1
C2
C3
C4
C5
$ git clone
Developer 1 (Local)
BugFix
B1
B2
B3
B4
B5
$ git pull
$ git merge
$ git push
Master (main) branch
C1
C2
C3
C4
C5
$ git pull
$ git merge
No conflicts
$ git merge
$ git push
$ git clone
Developer 2 (local)
Master (main) branch
C1
C2
C3
C4
$ git pull
$ git merge
Conflicts!
Sort conflicts
Test
A1
A2
A3
A4
A5
$ git pull
$ git merge
$ git push

Branching

Multiple branches can advance indipendantly



git is not perfect

- Complex information model
  - It is complicated – and you need to know all of it
- Crazy command line syntax
- Documentation isn’t good
- Burden of VCS maintenance pushed to contributors
- Unsafe version control

Rewrite history

git rebase

Rewrite history only if you have to

git rebase
git cherry-pick