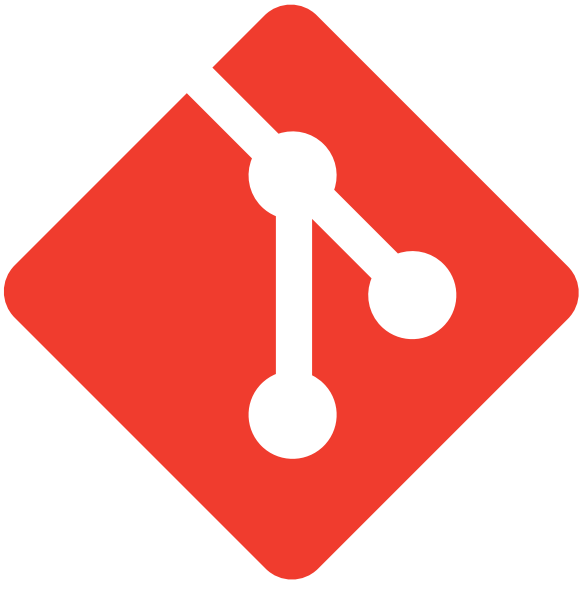


git

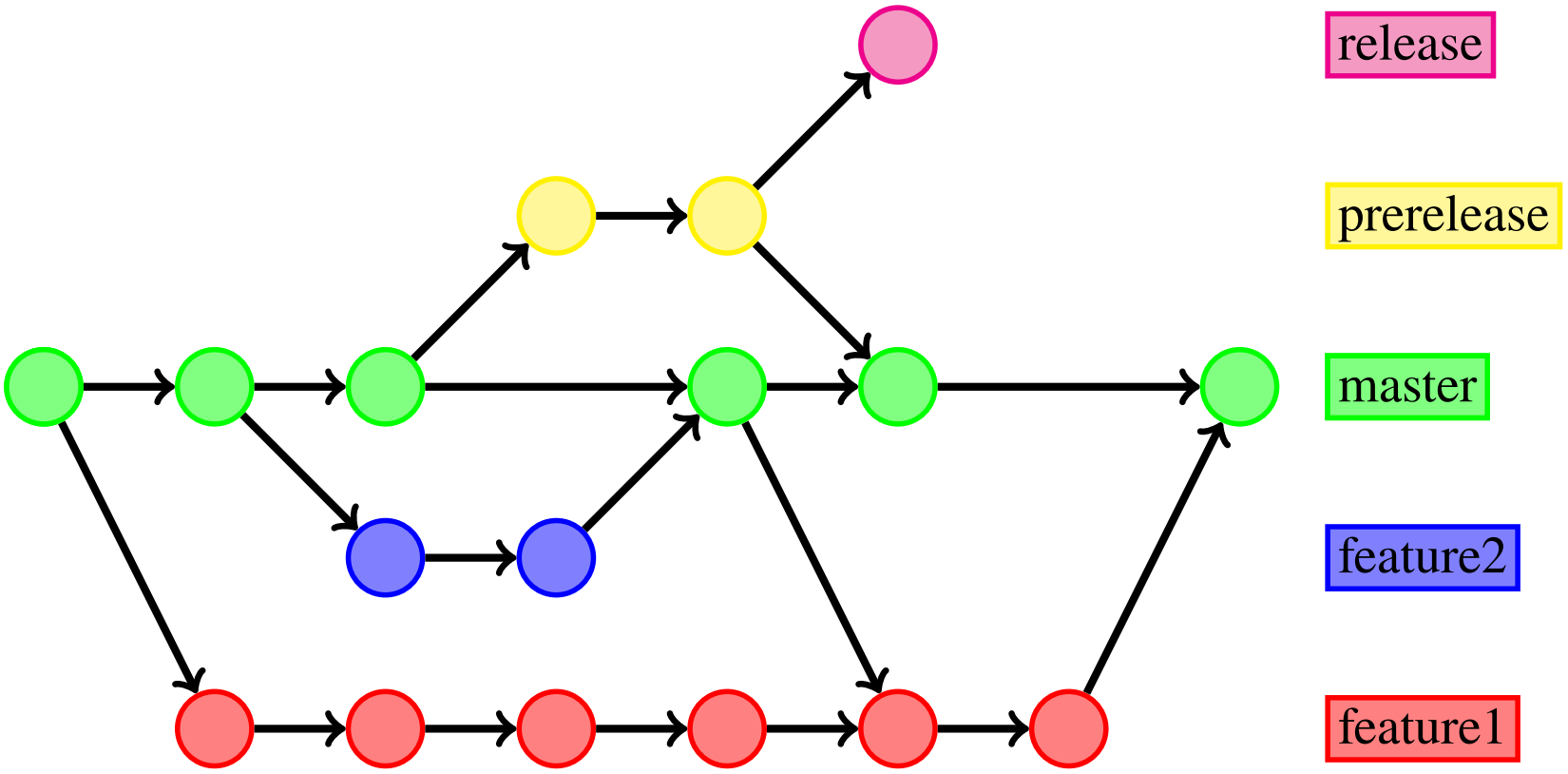
an introduction to the power of ”the stupid content tracker”

Arvid Conrad Ihrig

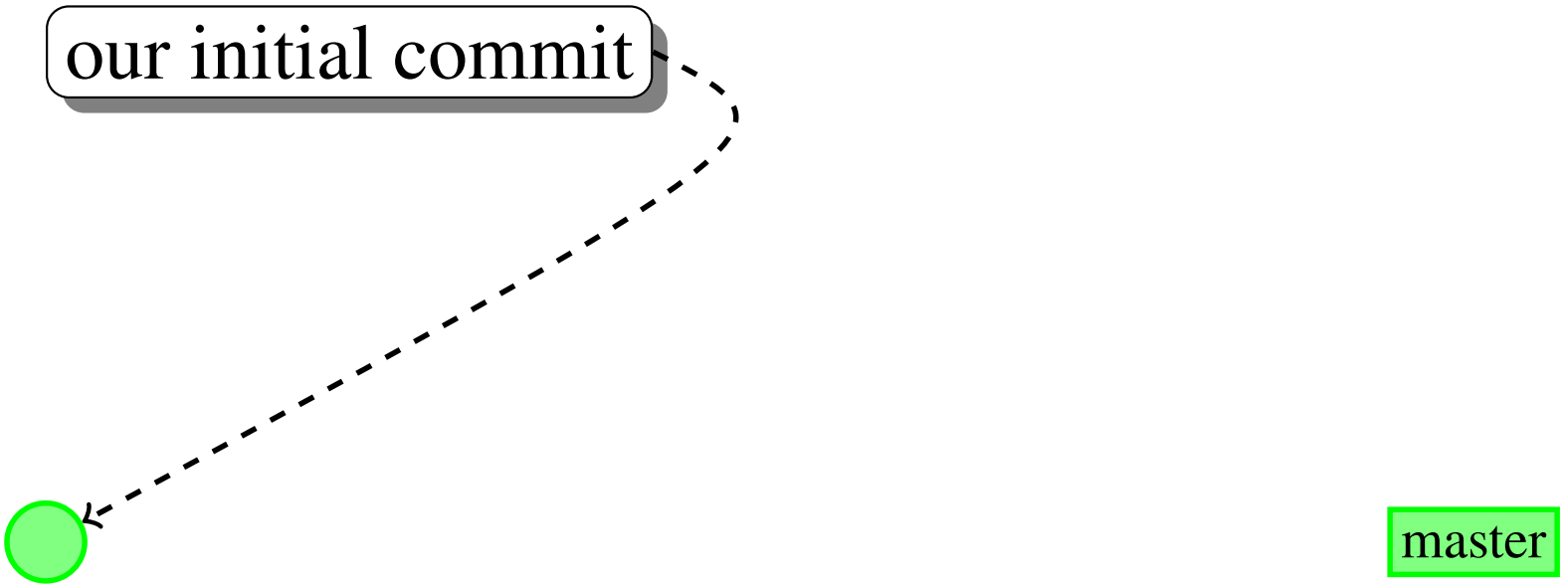
Fritz-Haber-Institut der Max-Planck-Gesellschaft

August 19th 2014, Berlin

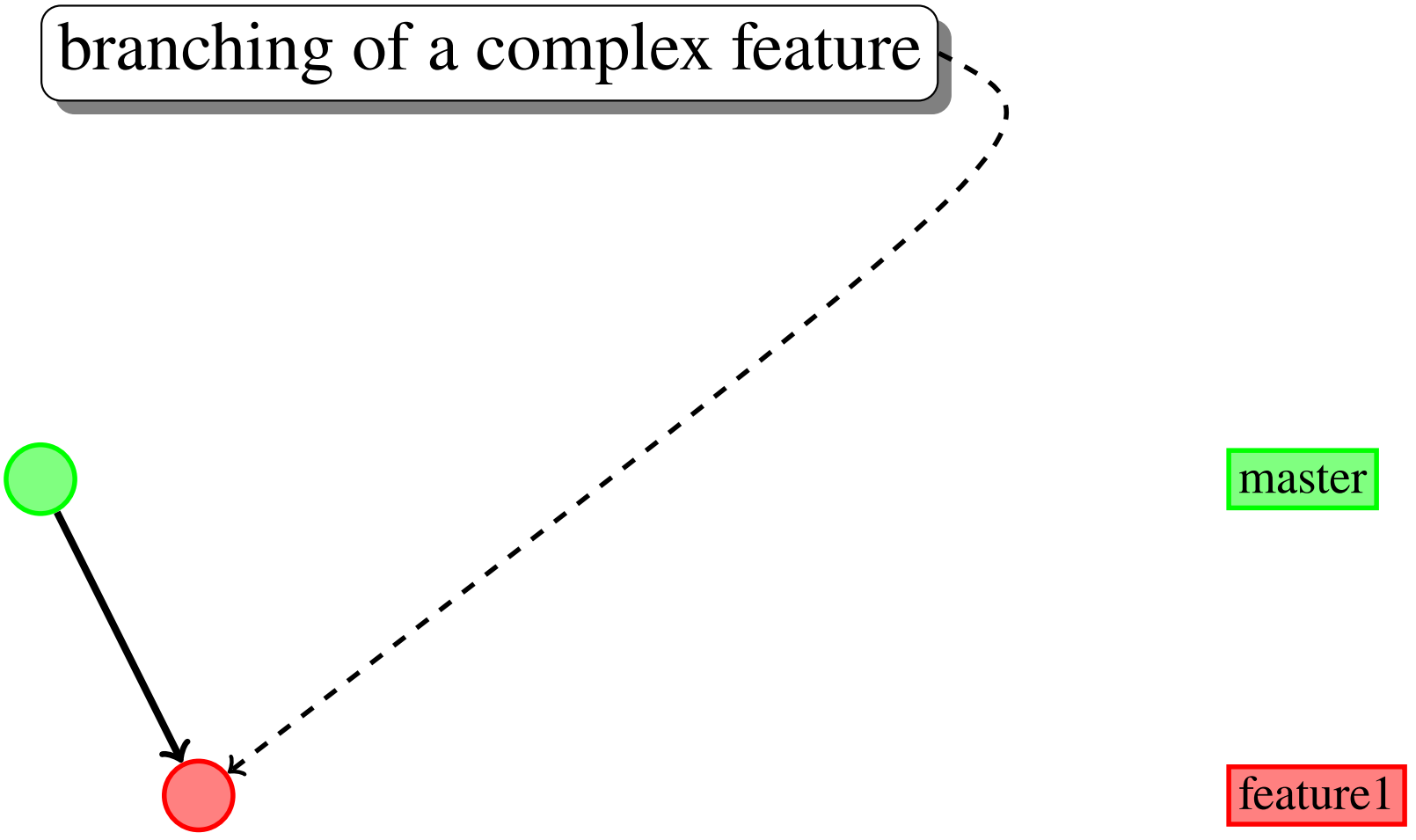
The Concept of Distributed Version Control



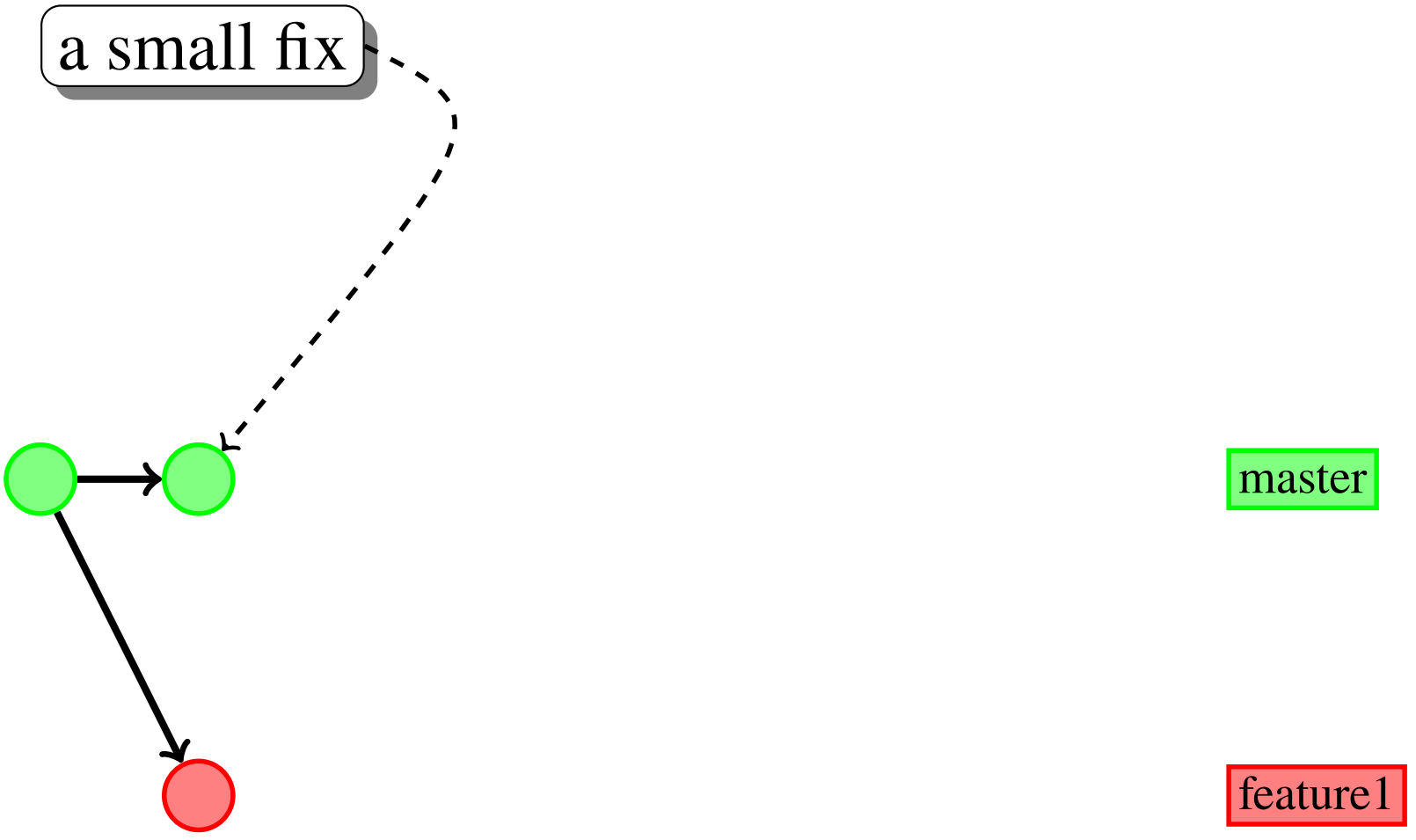
The Concept of Distributed Version Control



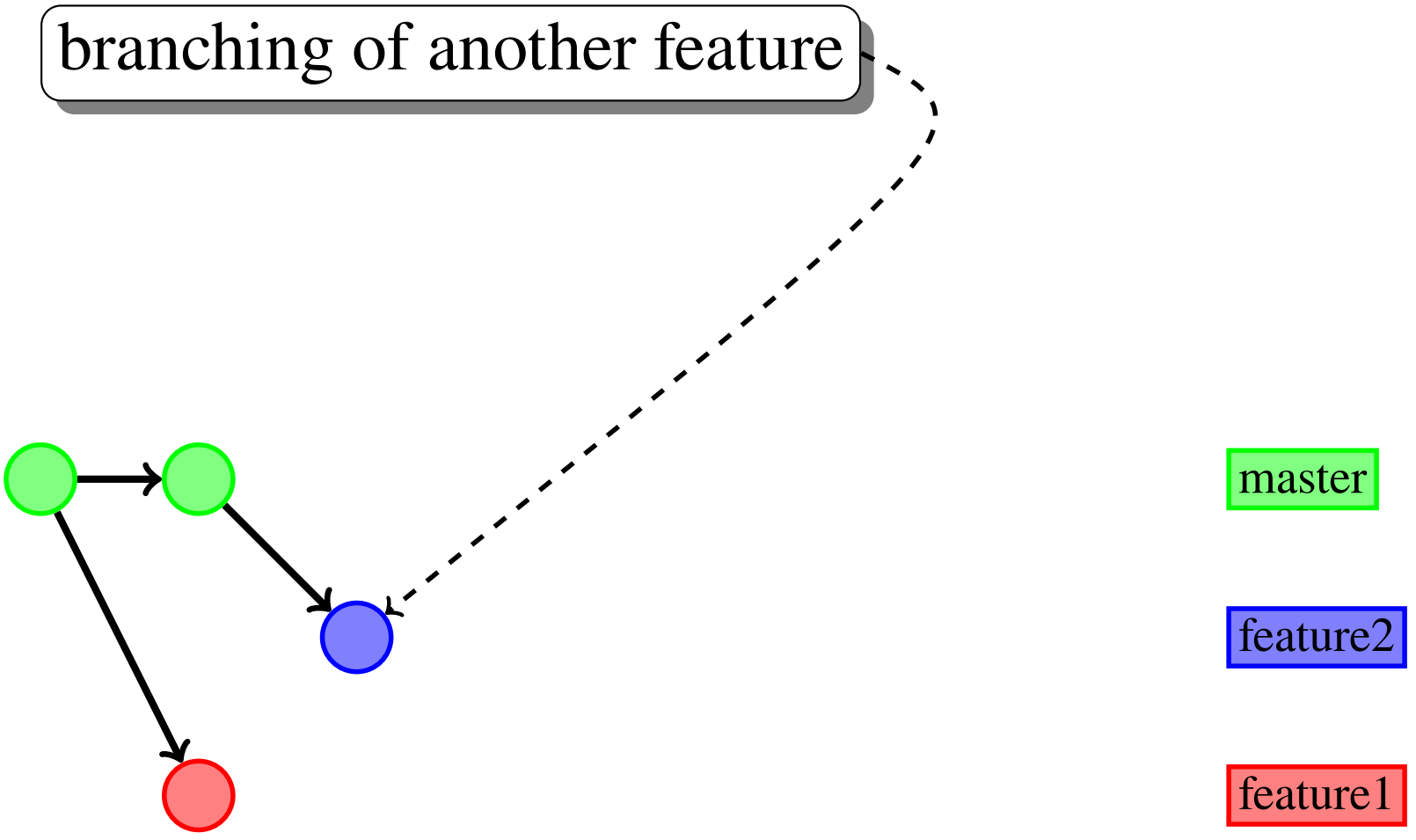
The Concept of Distributed Version Control



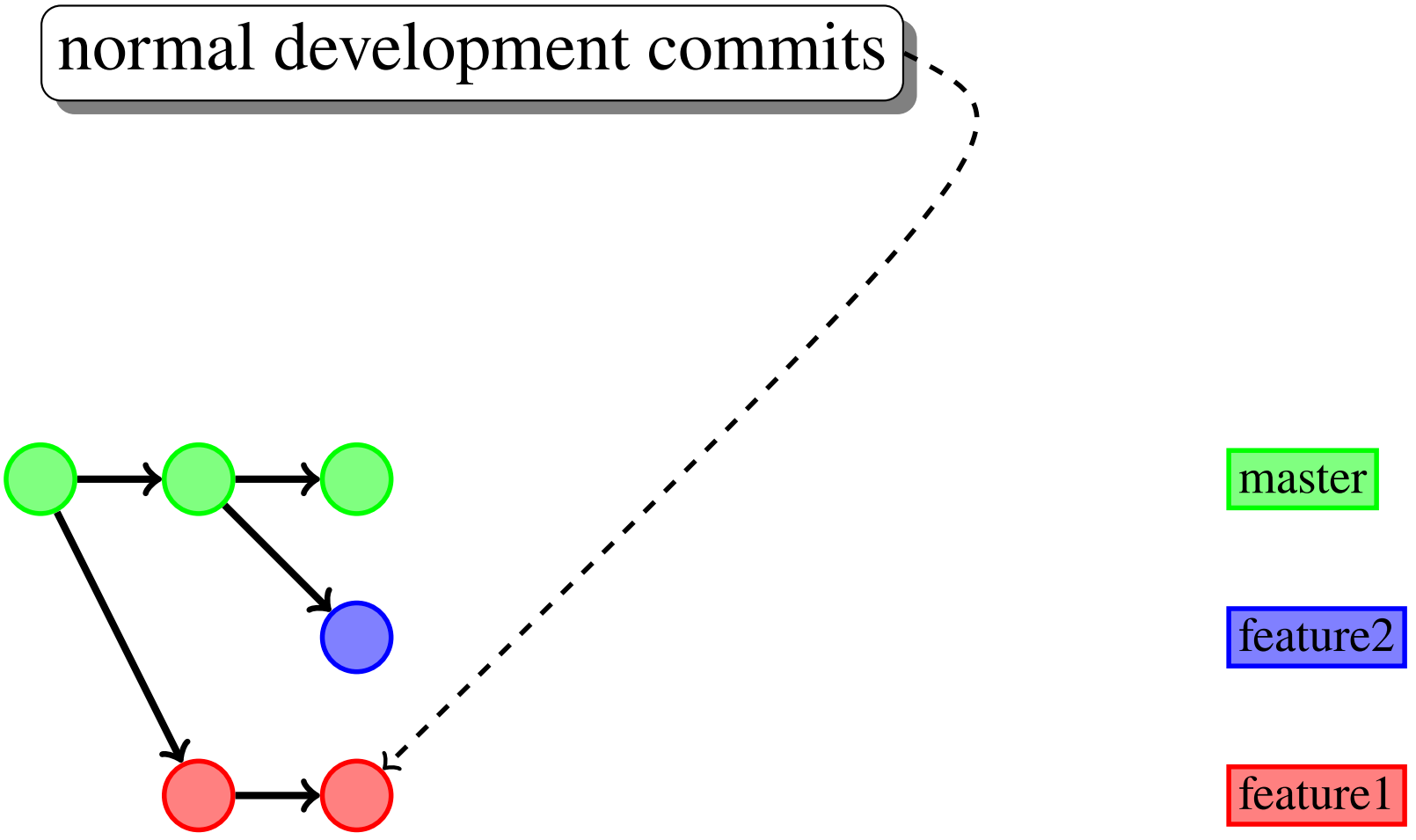
The Concept of Distributed Version Control



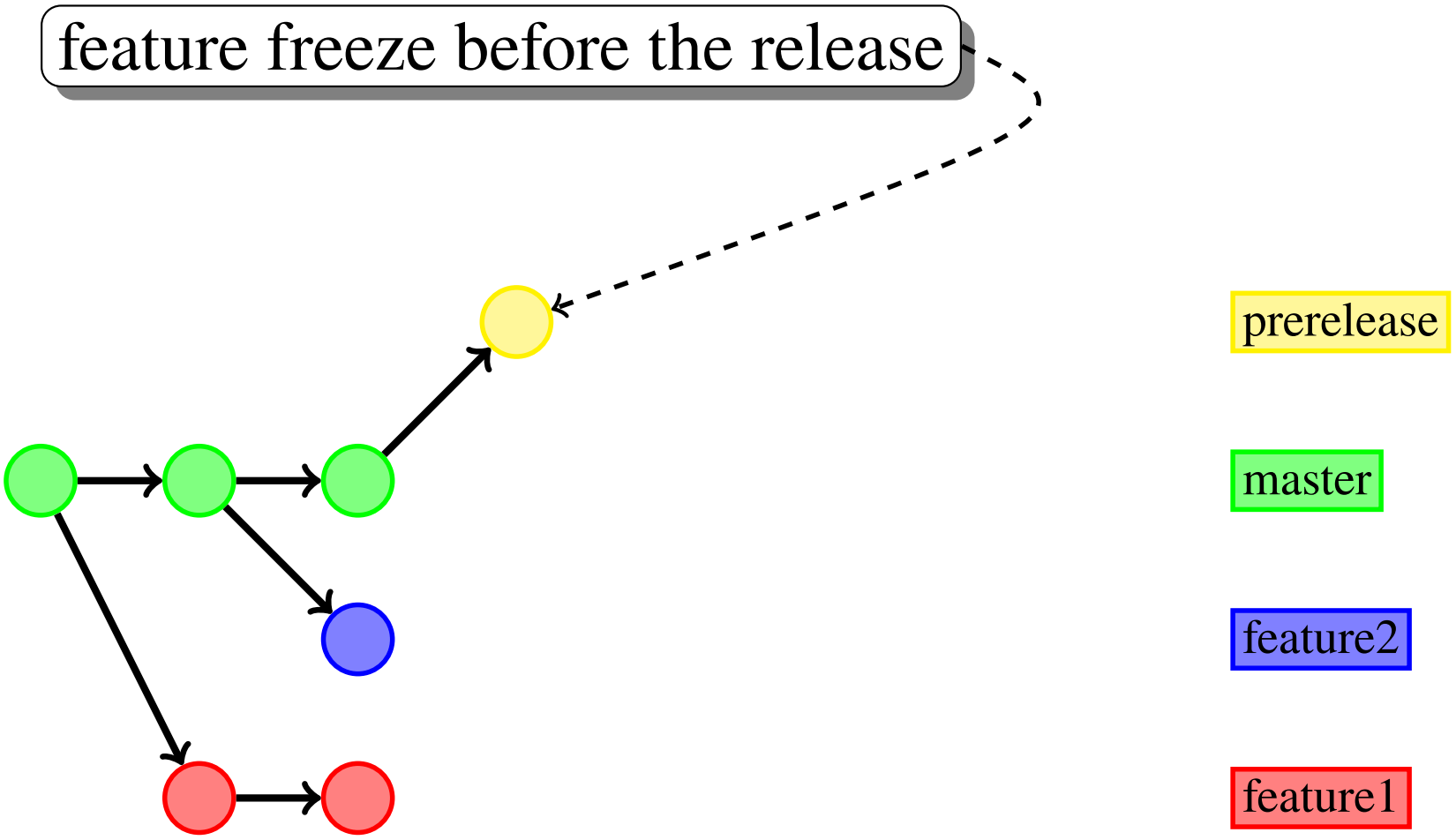
The Concept of Distributed Version Control



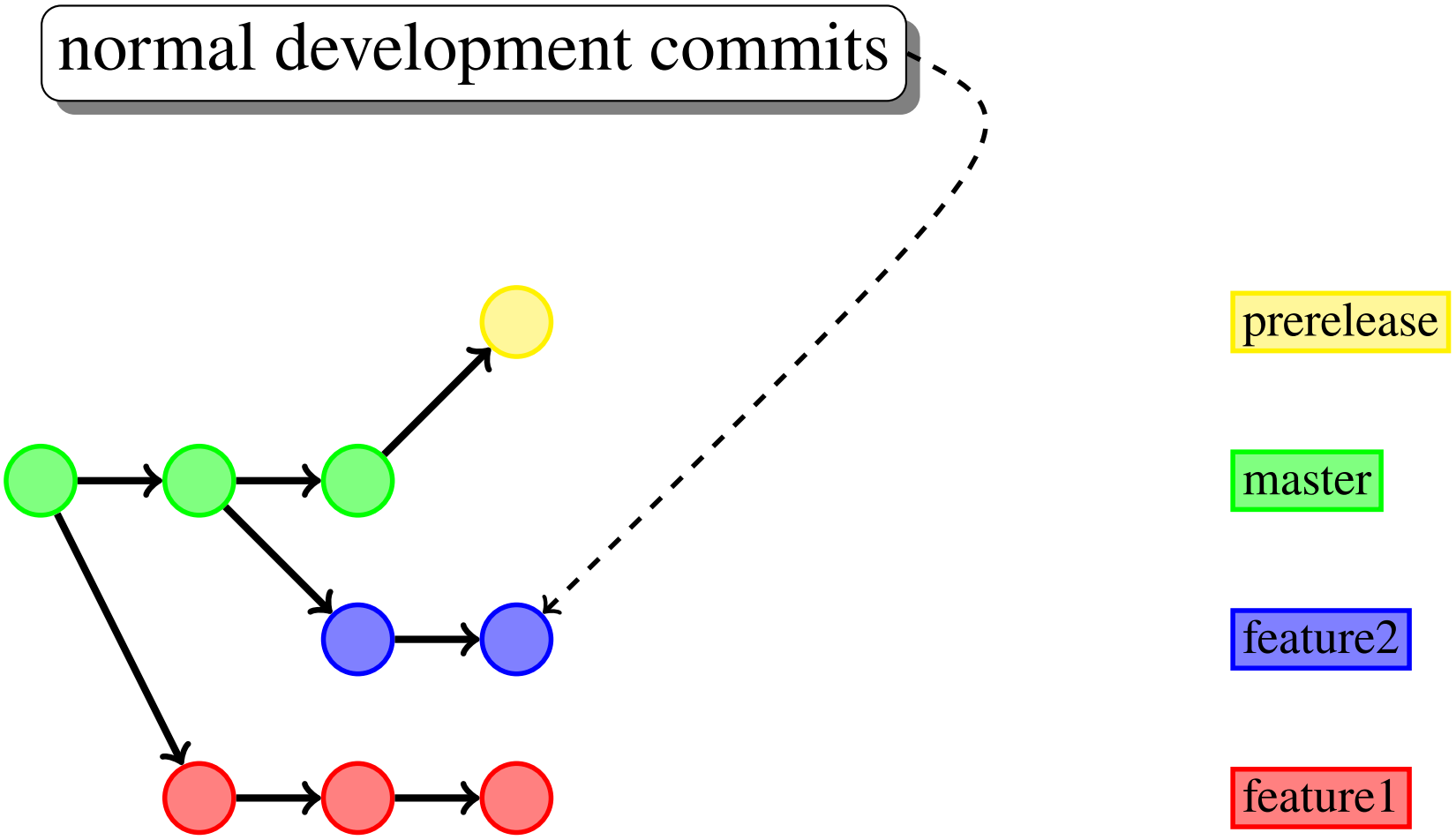
The Concept of Distributed Version Control



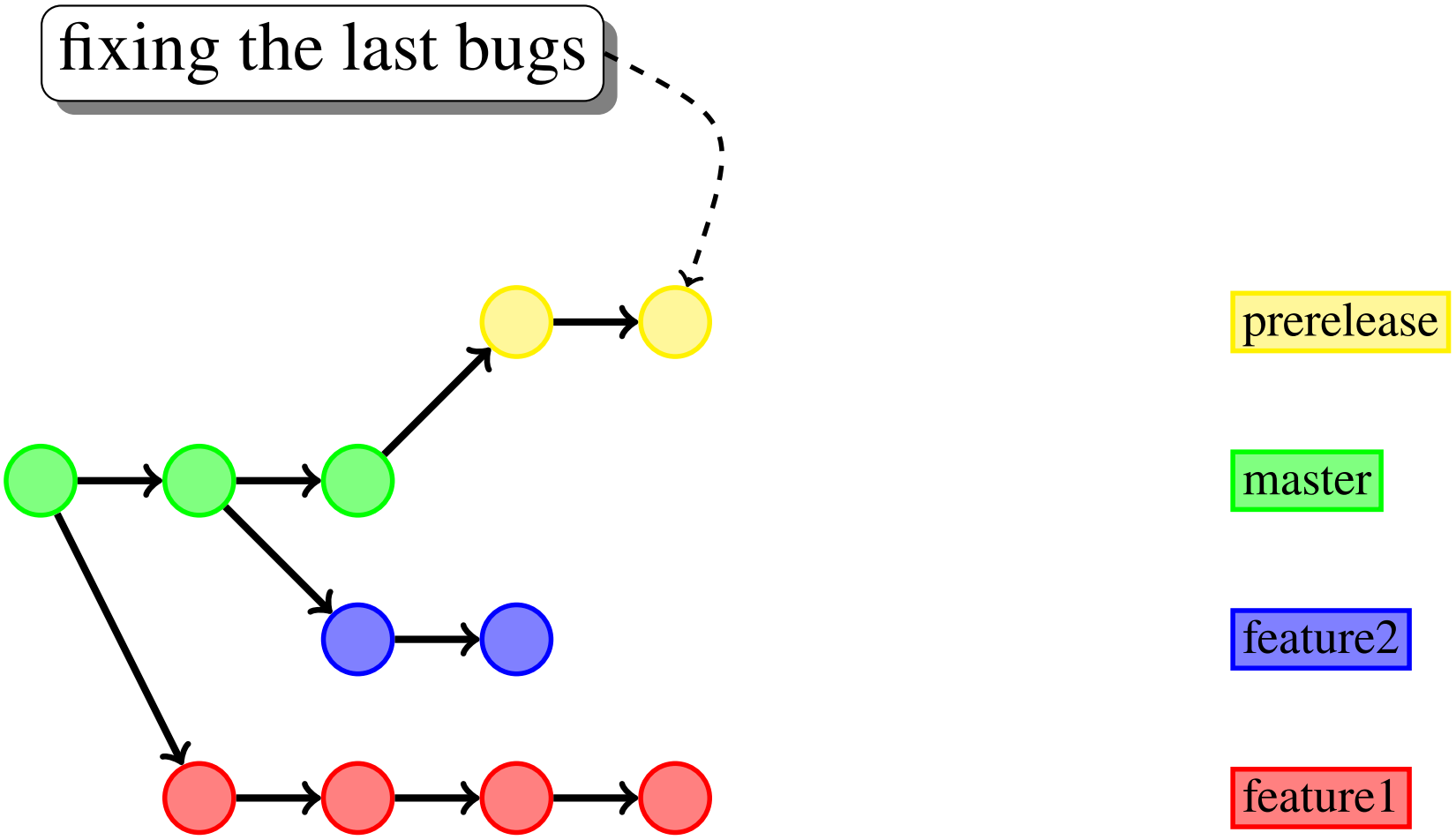
The Concept of Distributed Version Control



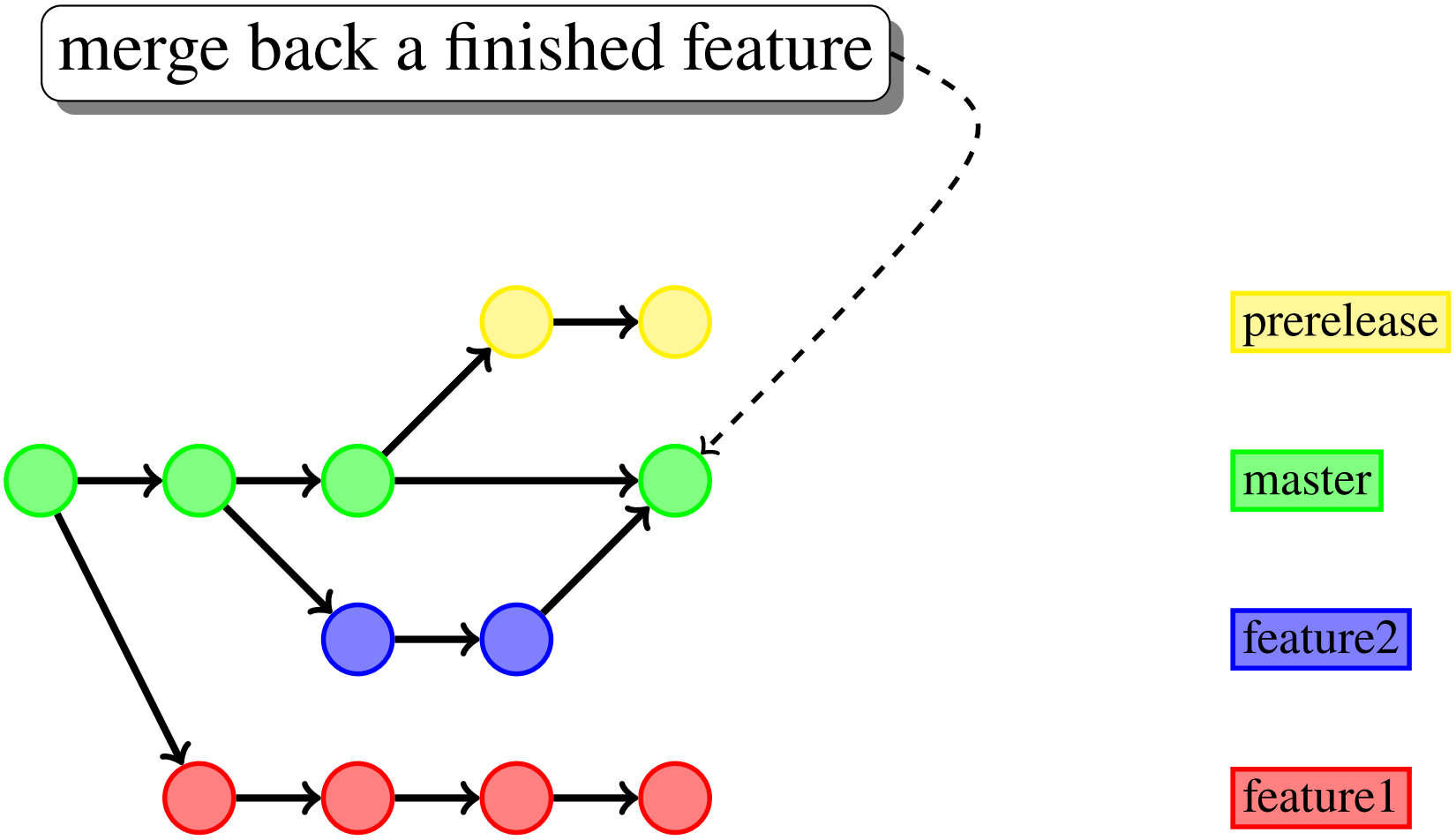
The Concept of Distributed Version Control



The Concept of Distributed Version Control

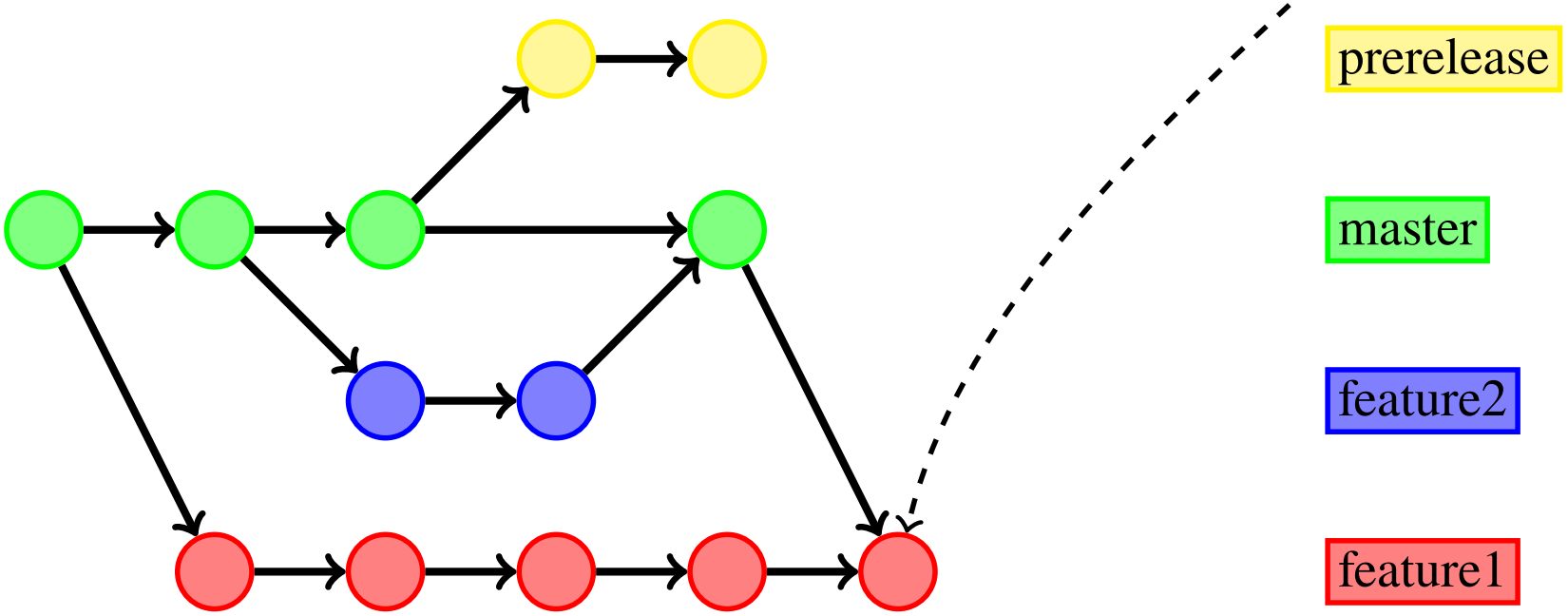


The Concept of Distributed Version Control

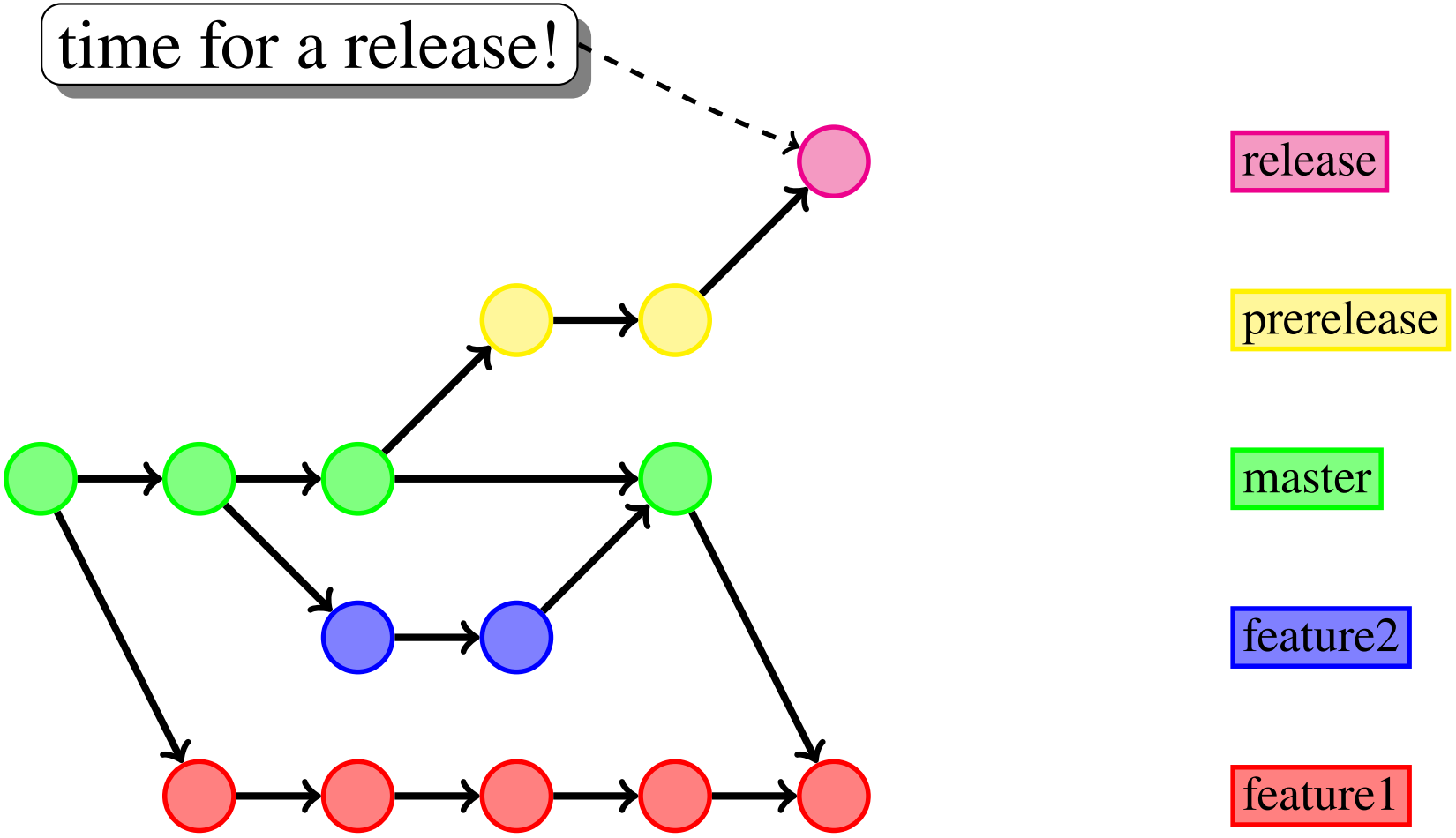


The Concept of Distributed Version Control

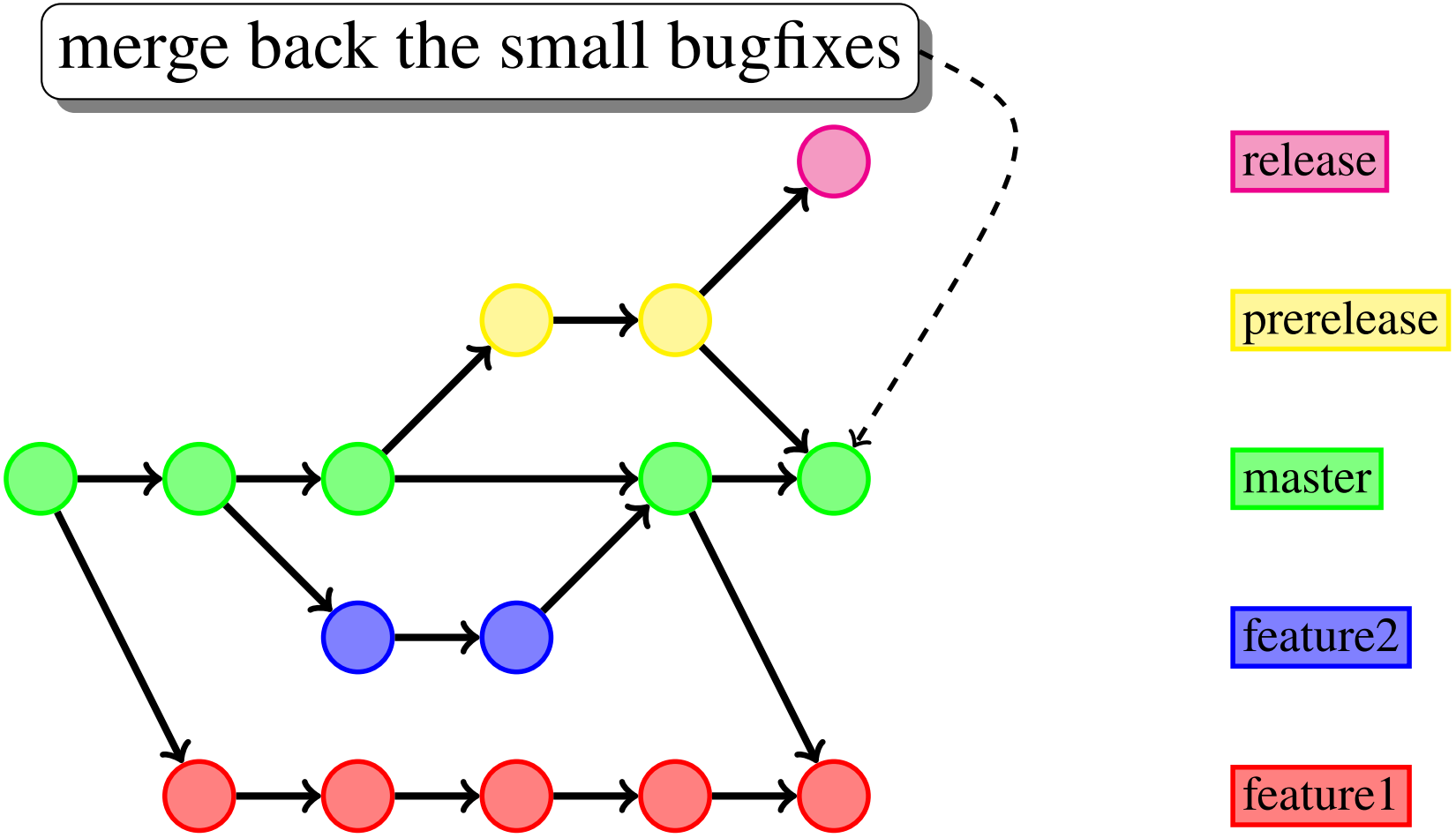
merge from master to keep feature up-to-date



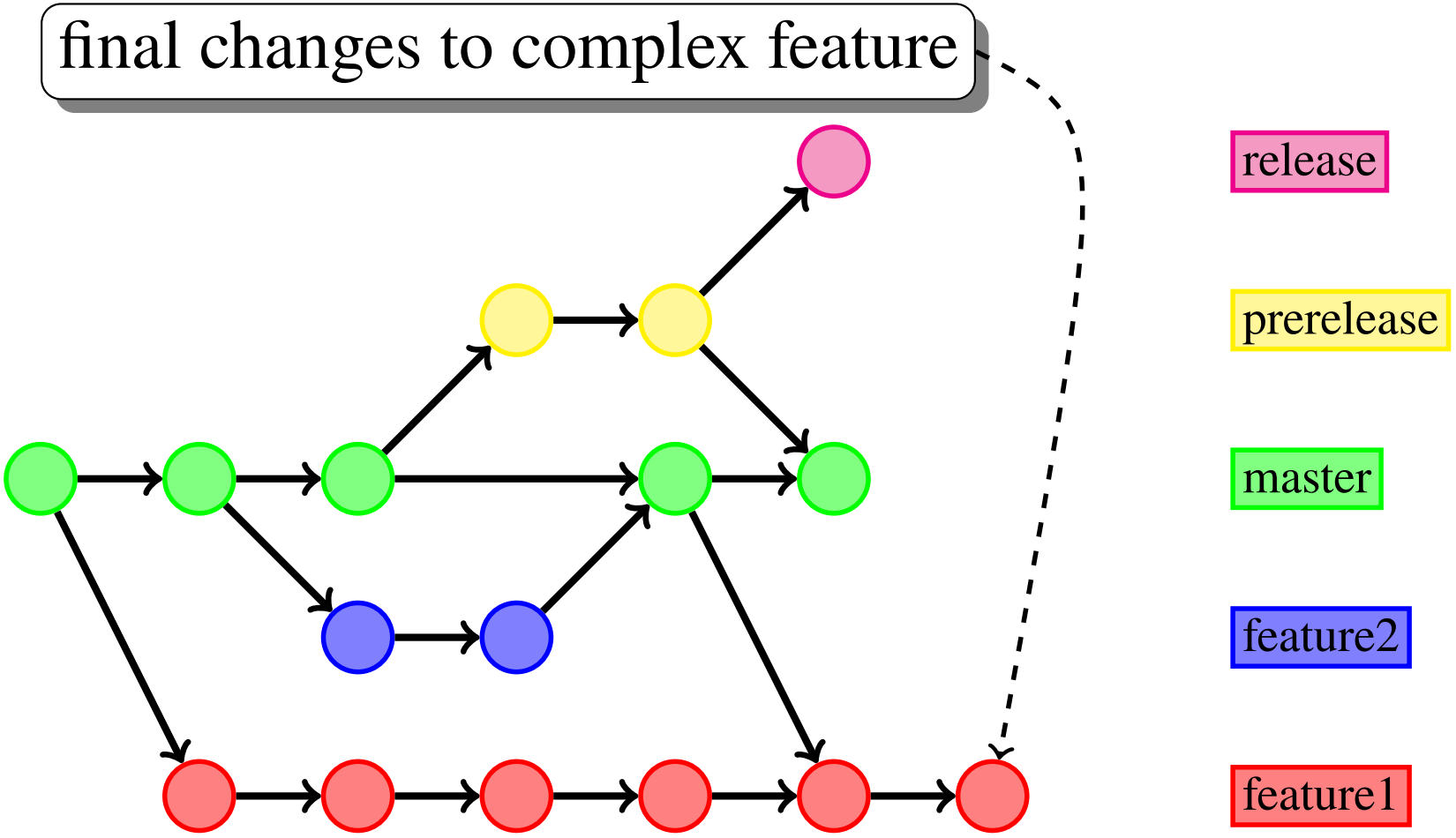
The Concept of Distributed Version Control



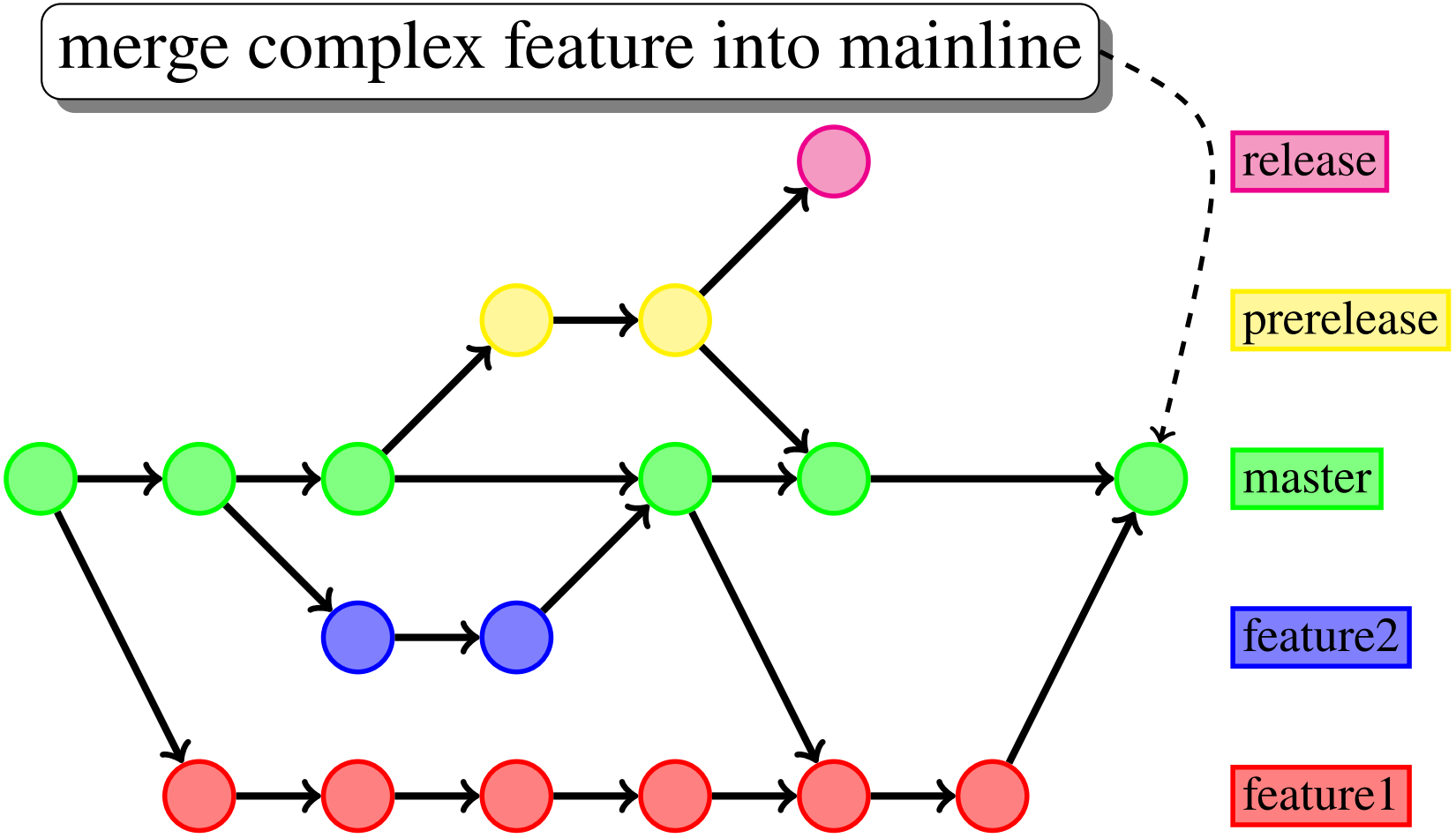
The Concept of Distributed Version Control



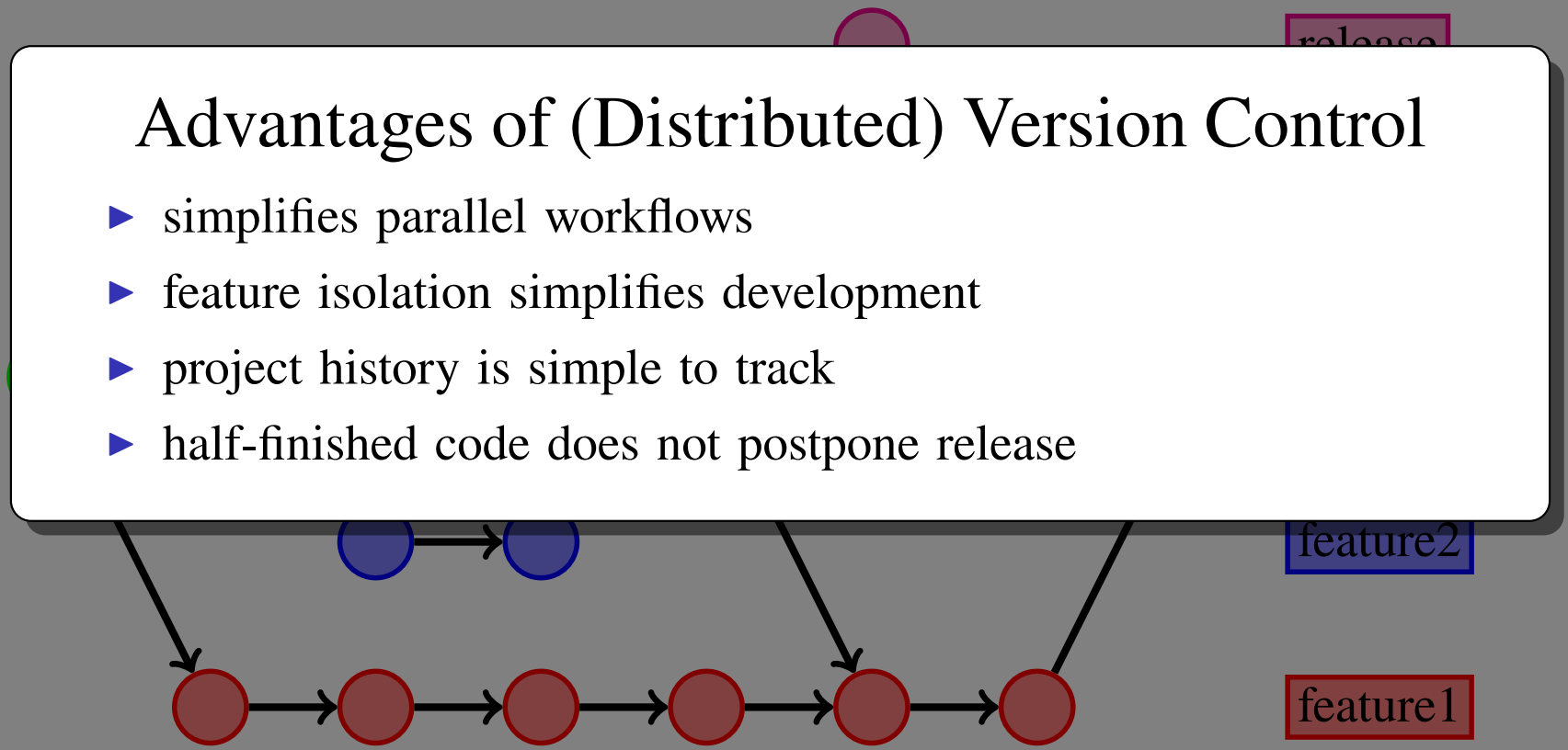
The Concept of Distributed Version Control



The Concept of Distributed Version Control



The Concept of Distributed Version Control



Advantages of (Distributed) Version Control

- simplifies parallel workflows
- feature isolation simplifies development
- project history is simple to track
- half-finished code does not postpone release

DVCS in git

A few general remarks about git:

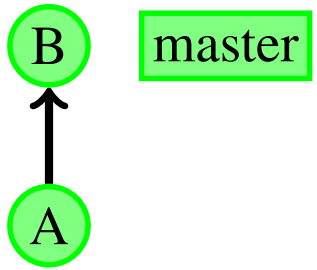
- each commit is **immutable** and uniquely identified by a checksum
  - commit ”modifying” operations create a new commit and discard the old one
  - do not modify already pushed commits
- git knows *local* and *remote* branches
  - branches are just *pointers* to specific commits
  - local branches are those you work on
  - remote branches indicate the state of the remote repository
- git can interact with more than one remote repository
  - for simplicity, we consider only one in this tutorial

Basic git usage
recording changes, inspecting commits and remote repositories

Cloning an existing git repository

Local Repository

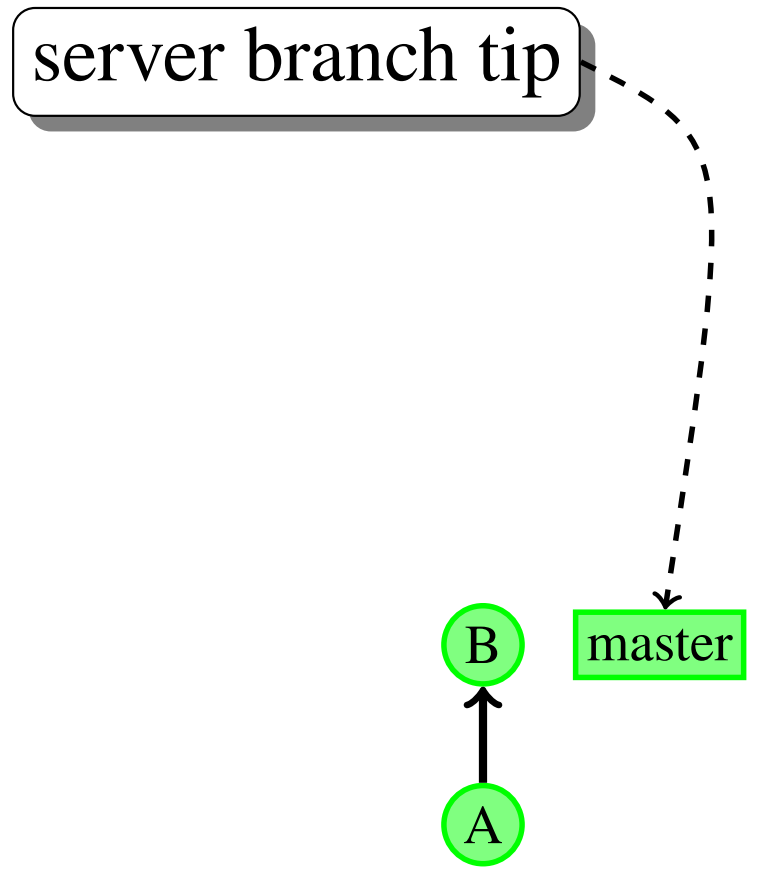
Remote Repository



Cloning an existing git repository

Local Repository

Remote Repository



Cloning an existing git repository

Local Repository | Remote Repository

`git clone <source> [<target>]`

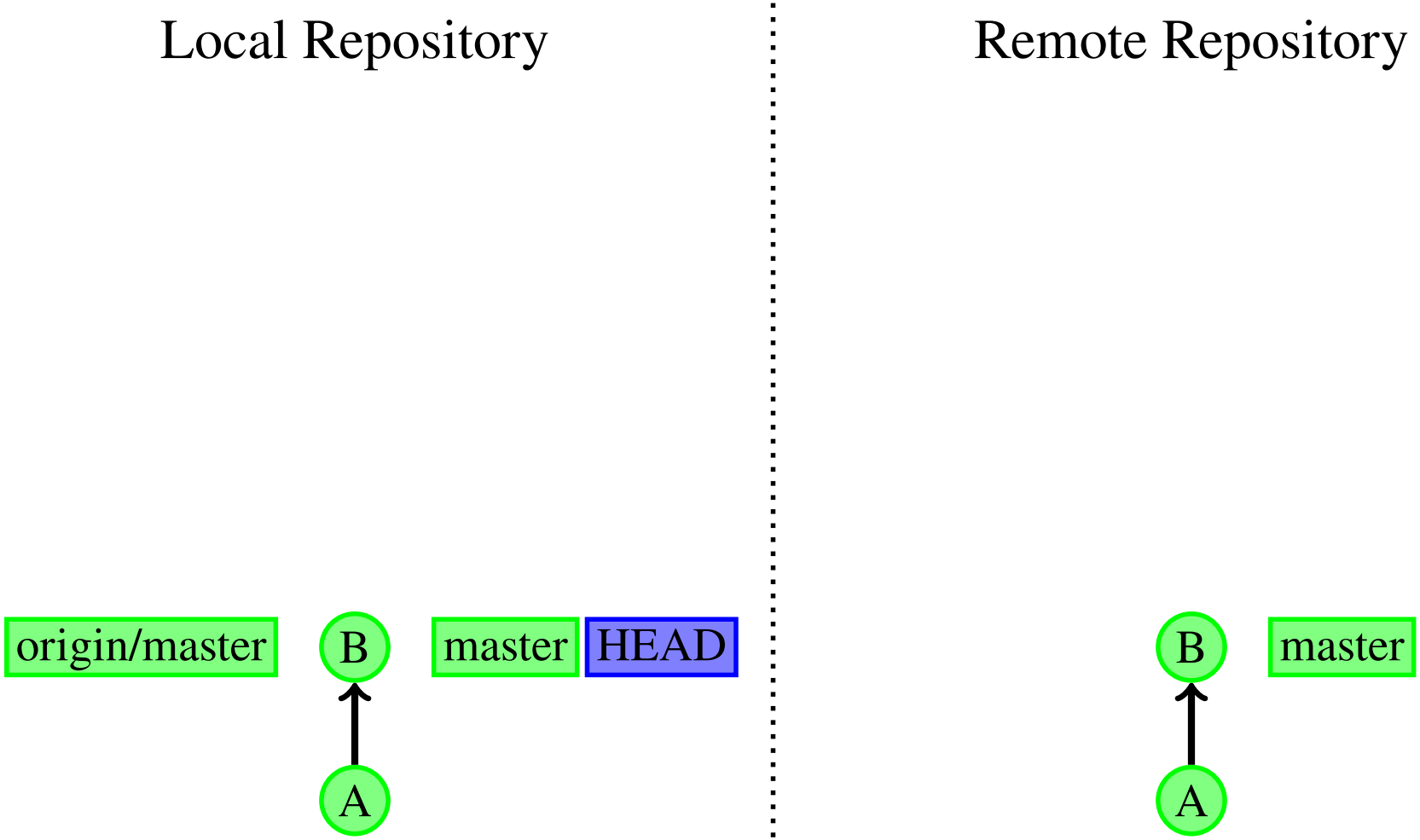
make a copy of the specified git repository

`source` any git repository, either from a local network or the web

`target` the (non-existing) folder into which the clone will be copied

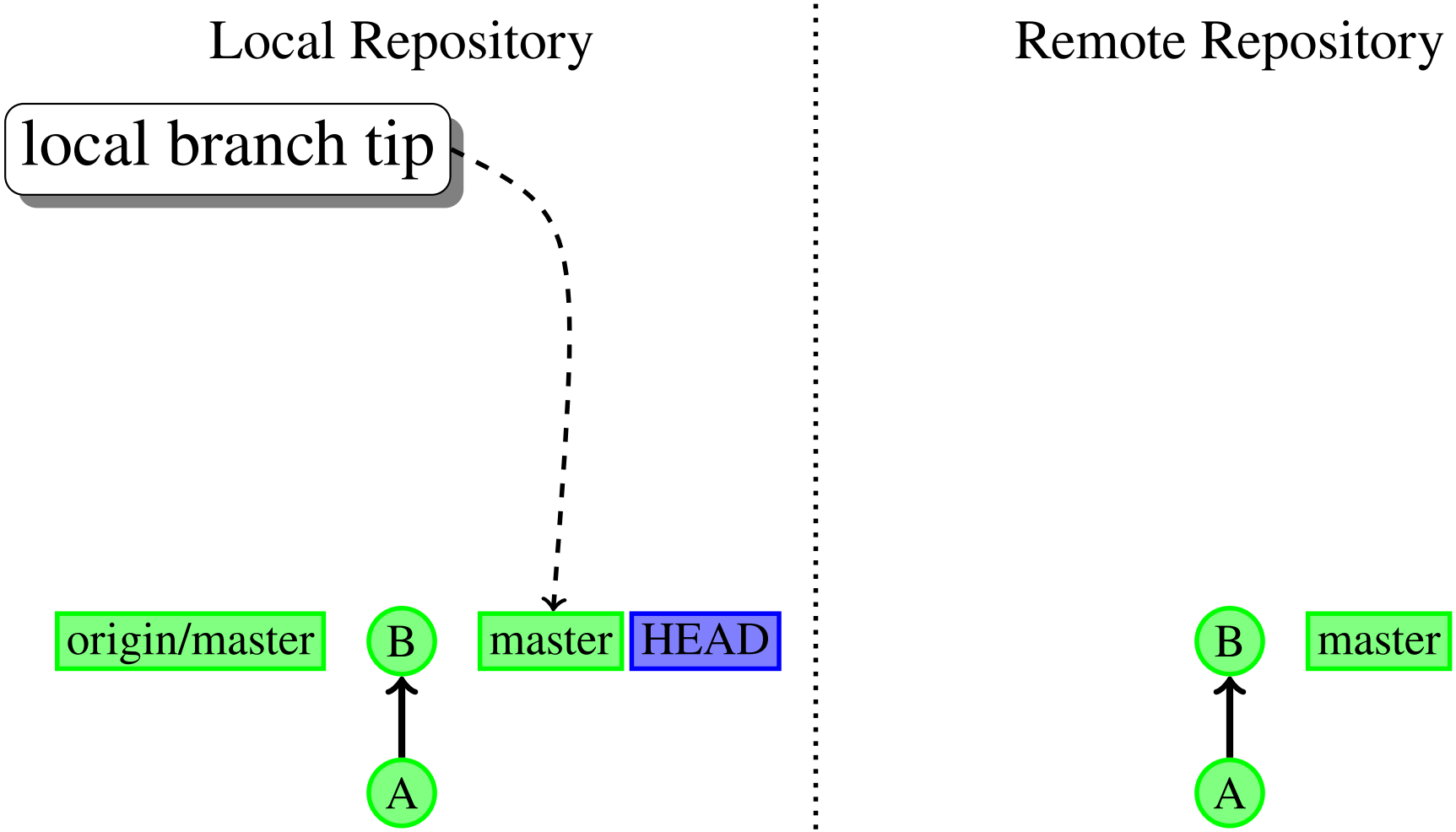

```
git@test: git clone user@server:path/to/repo
```

Cloning an existing git repository



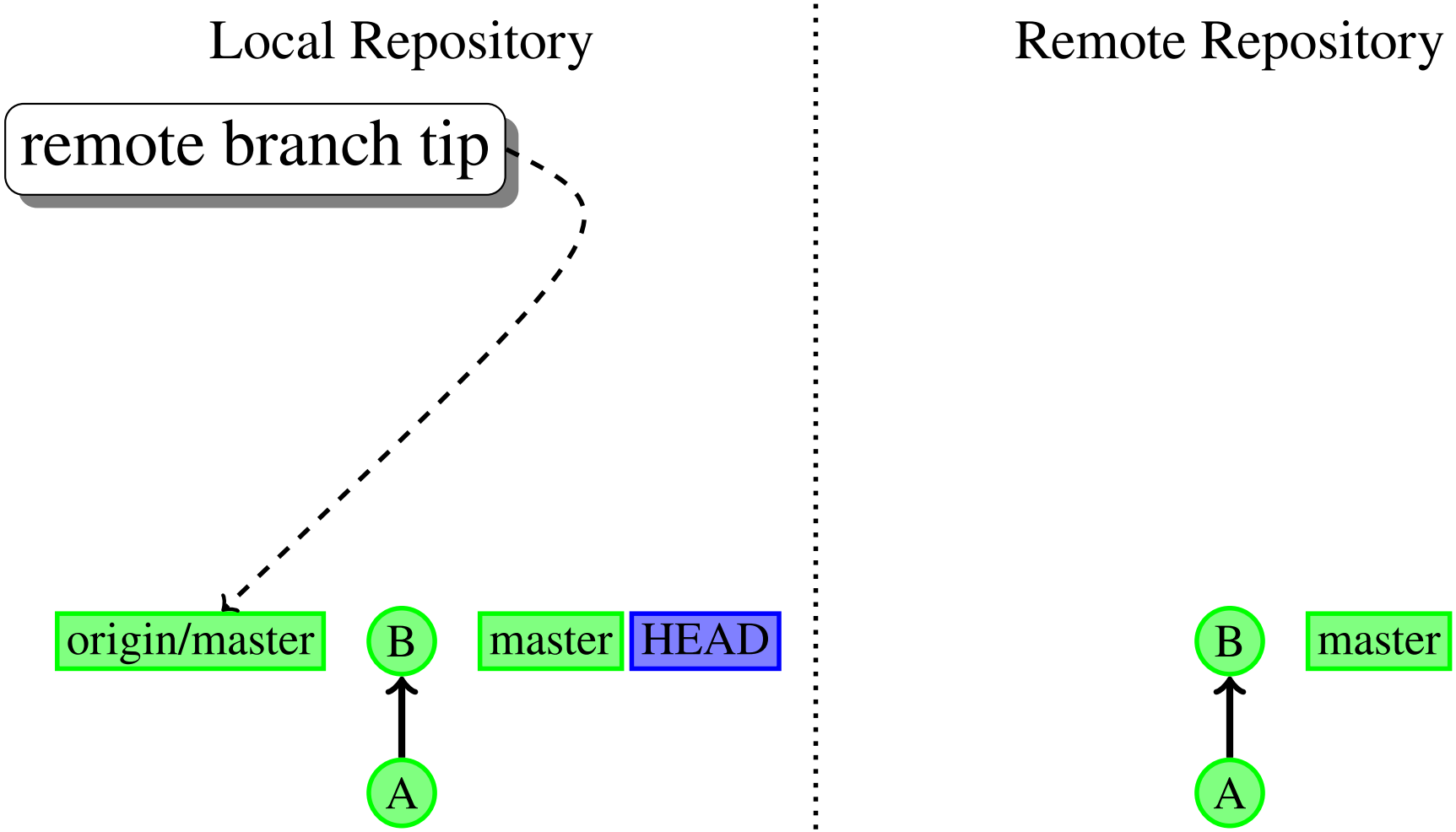

```
git@test: git clone user@server:path/to/repo
```

Cloning an existing git repository



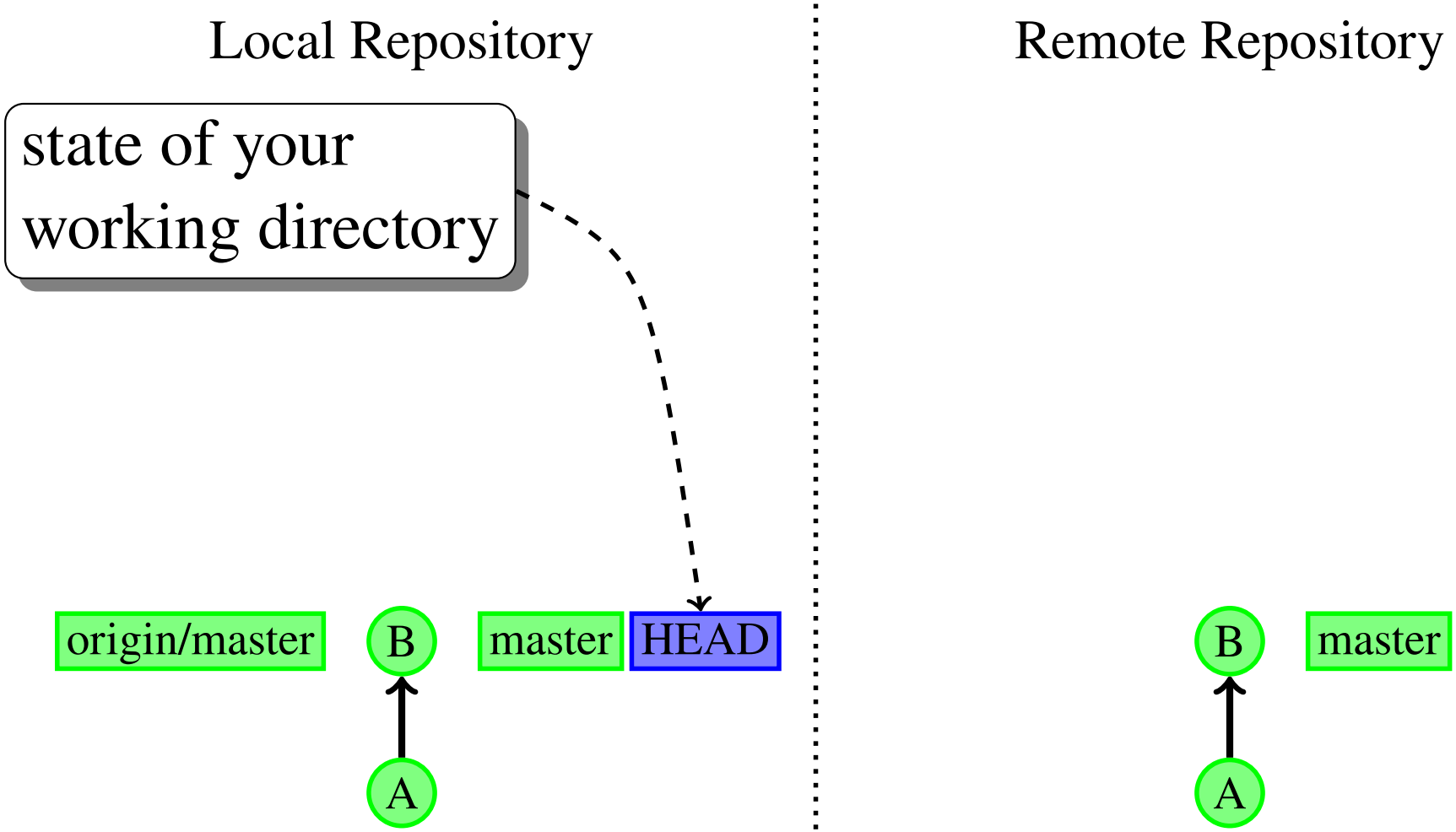

```
git@test: git clone user@server:path/to/repo
```

Cloning an existing git repository



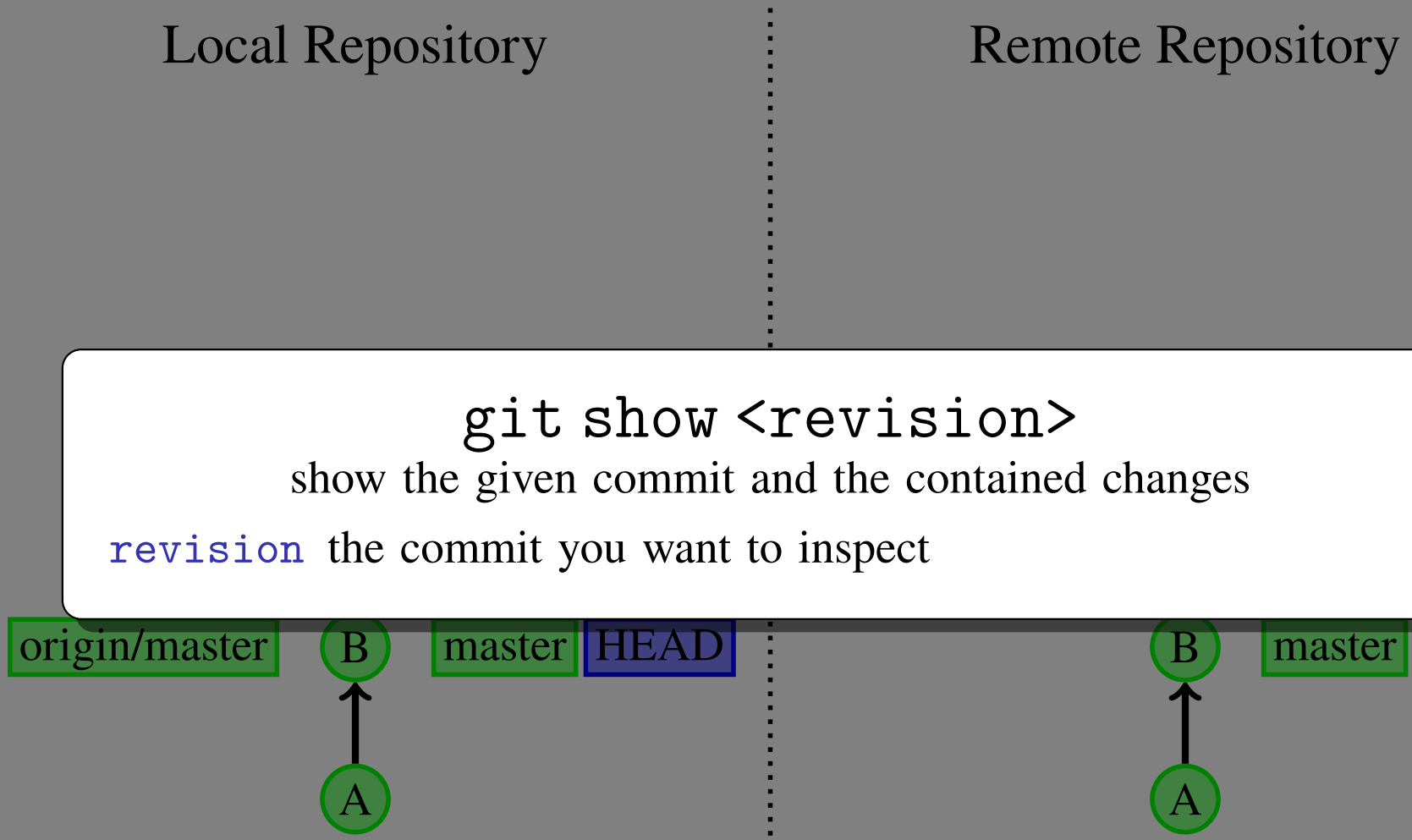

```
git@test: git clone user@server:path/to/repo
```

Cloning an existing git repository



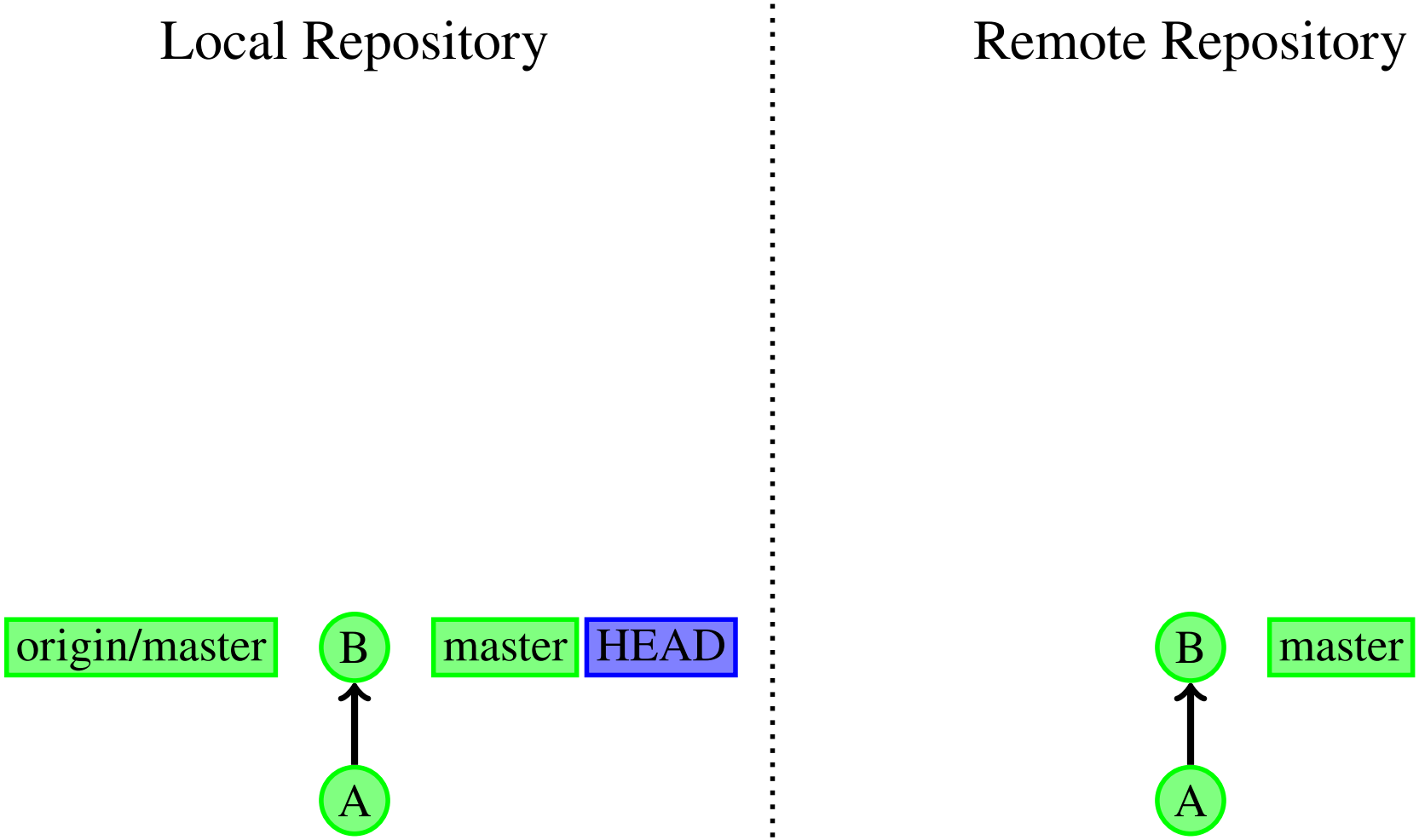

```
git@test: git clone user@server:path/to/repo
```

Cloning an existing git repository




```
git@test: git clone user@server:path/to/repo
```

Committing your changes locally




```
git@test: echo "This should be fun!" > newfile.txt
```

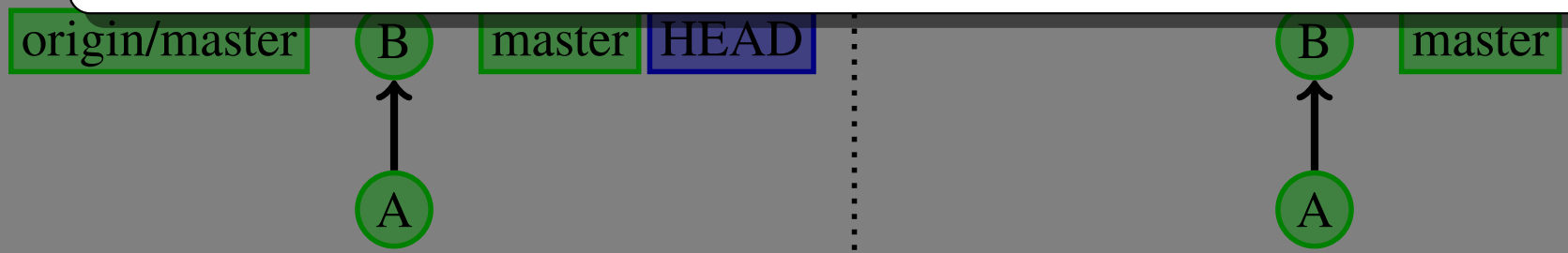
Committing your changes locally

Local Repository | Remote Repository

`git status [<path>]`

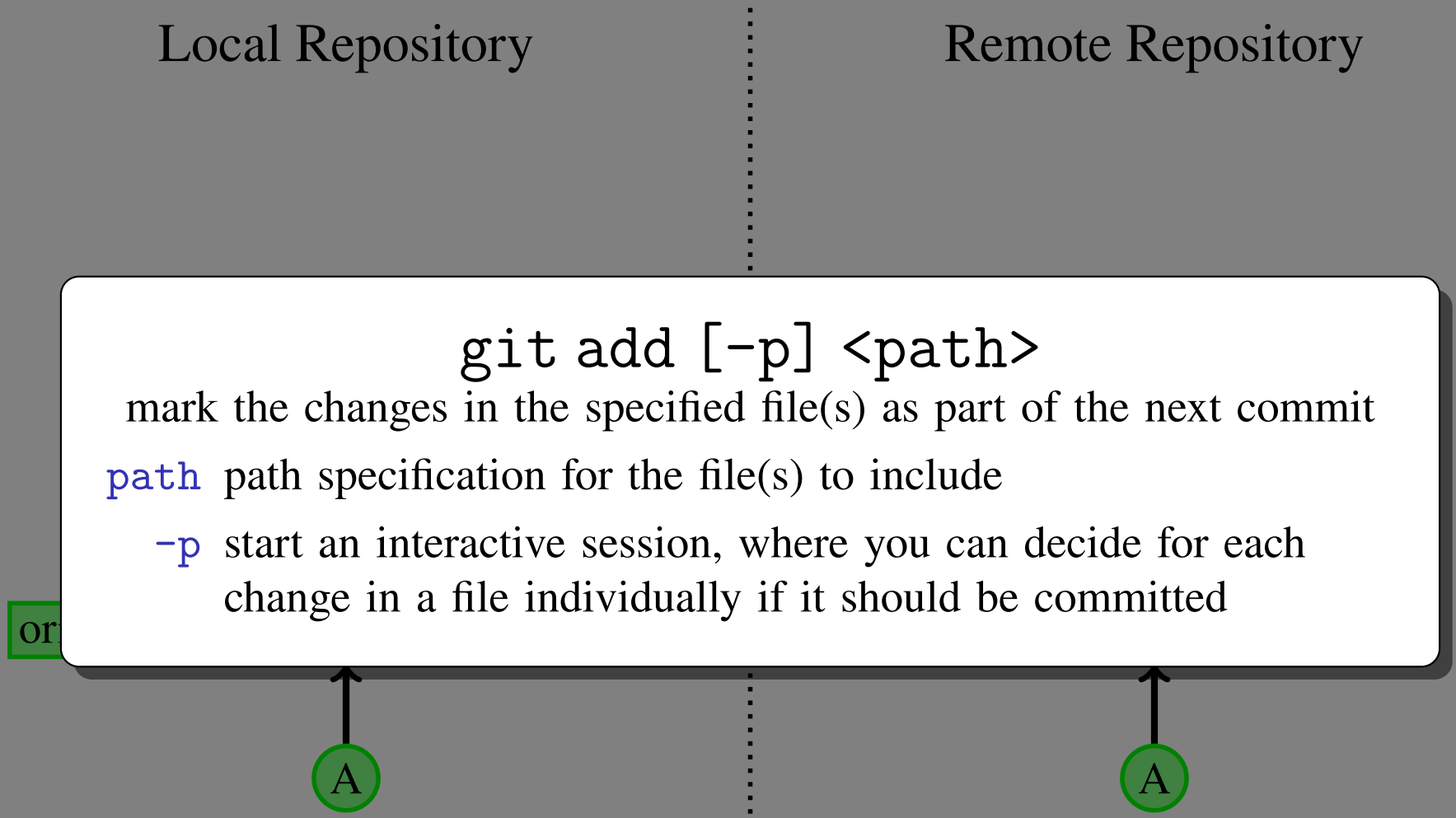
show what changed compared to the current HEAD revision

`path` restrict output to files and folders matching this shell pattern



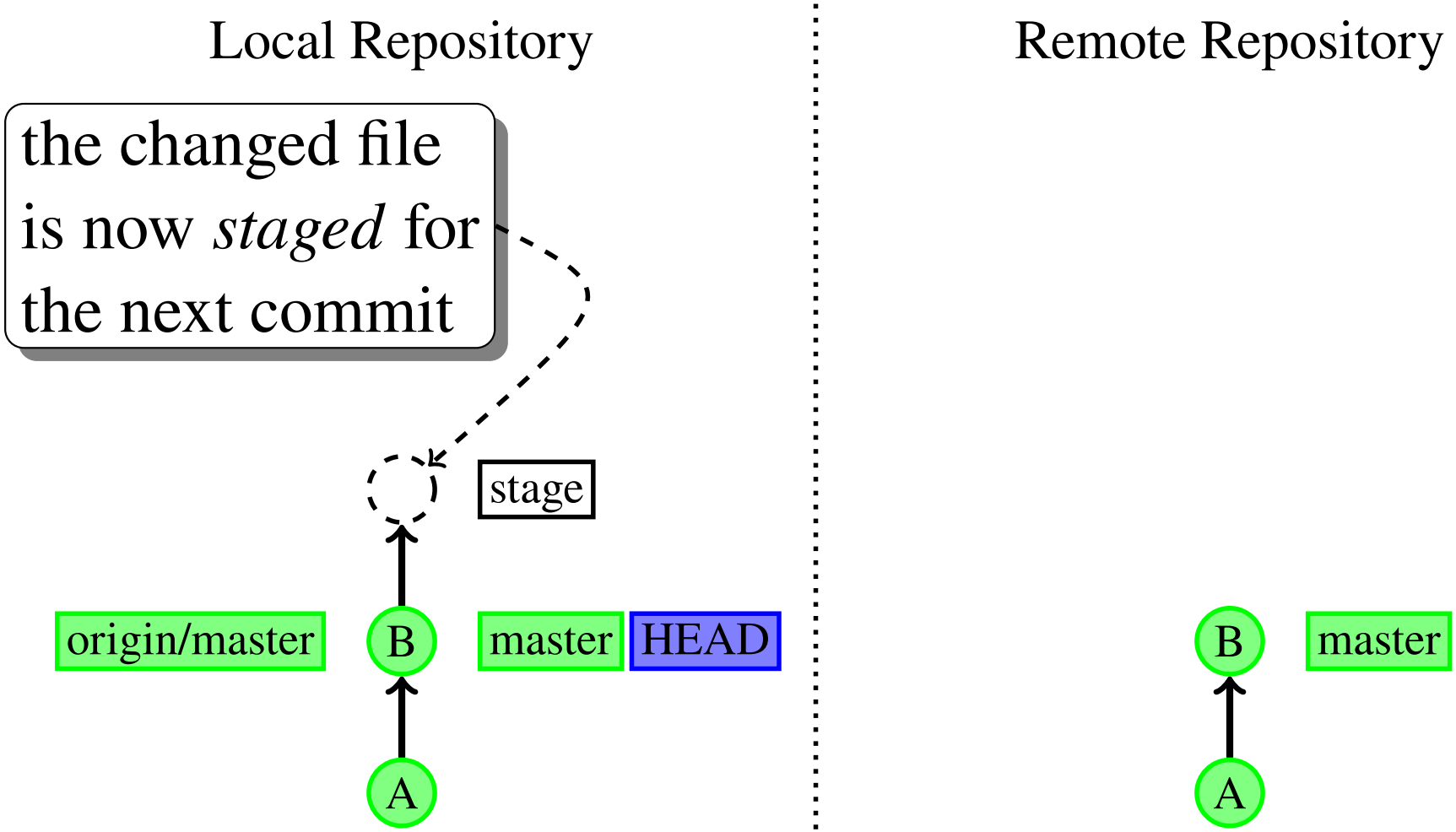

```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
```

Committing your changes locally



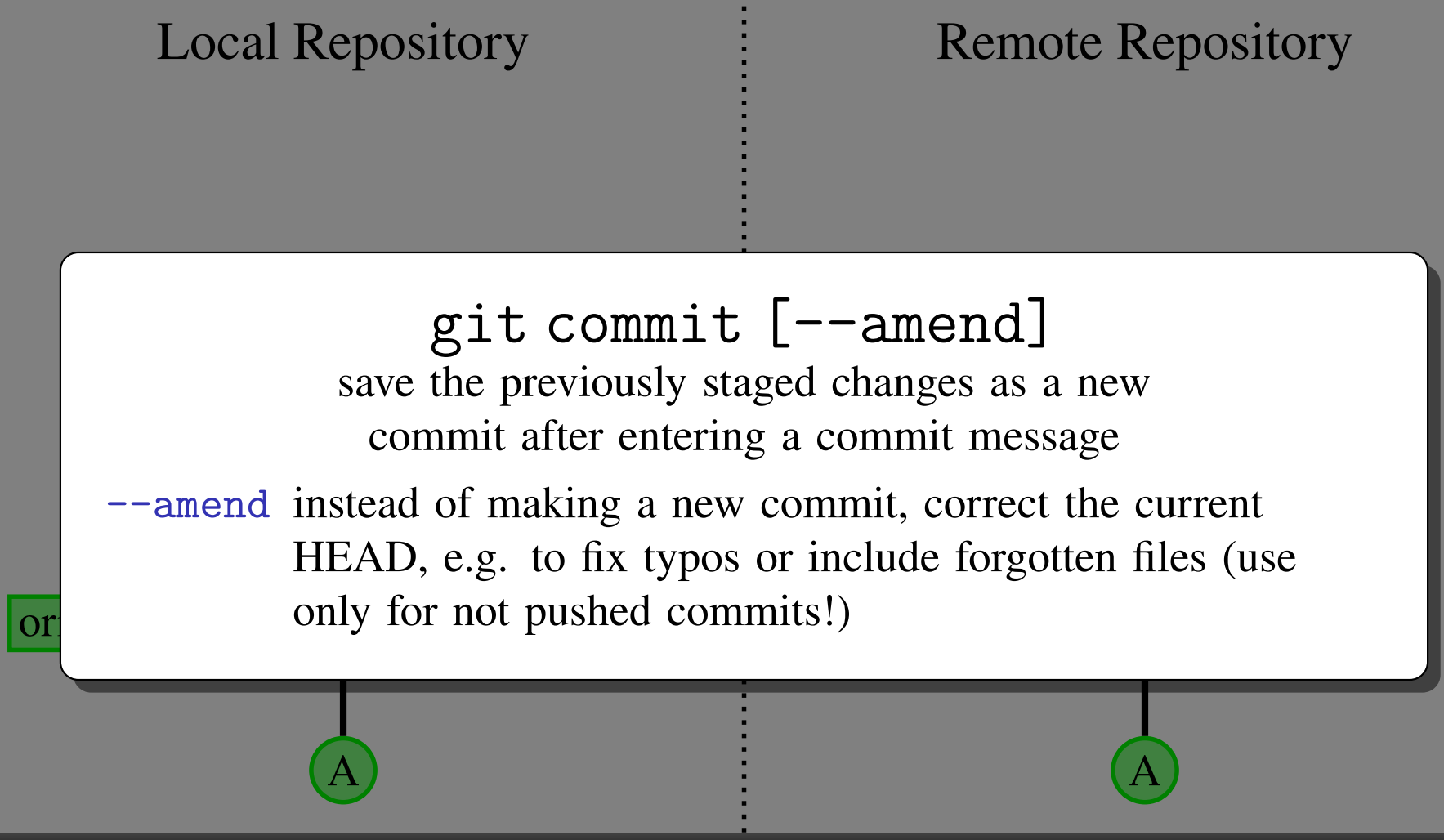

```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
git@test: git add newfile.txt
```

Committing your changes locally




```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
git@test: git add newfile.txt
```

Committing your changes locally

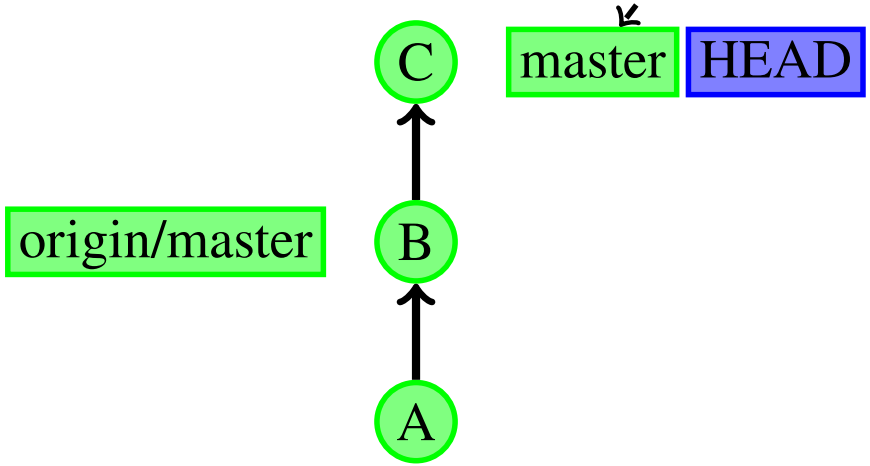



```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
git@test: git add newfile.txt
git@test: git commit
```

Committing your changes locally

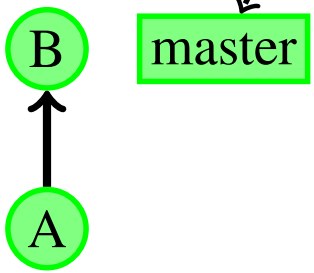
Local Repository

only the local branch has changed



Remote Repository

git commit changes nothing on the server




```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
git@test: git add newfile.txt
git@test: git commit
```

Committing your changes locally

Local Repository | Remote Repository

`git diff [--cached | <revision1>] [<revision2>] <path>`

show a diff for a file between two different revisions,
with no arguments diffs working directory vs. HEAD

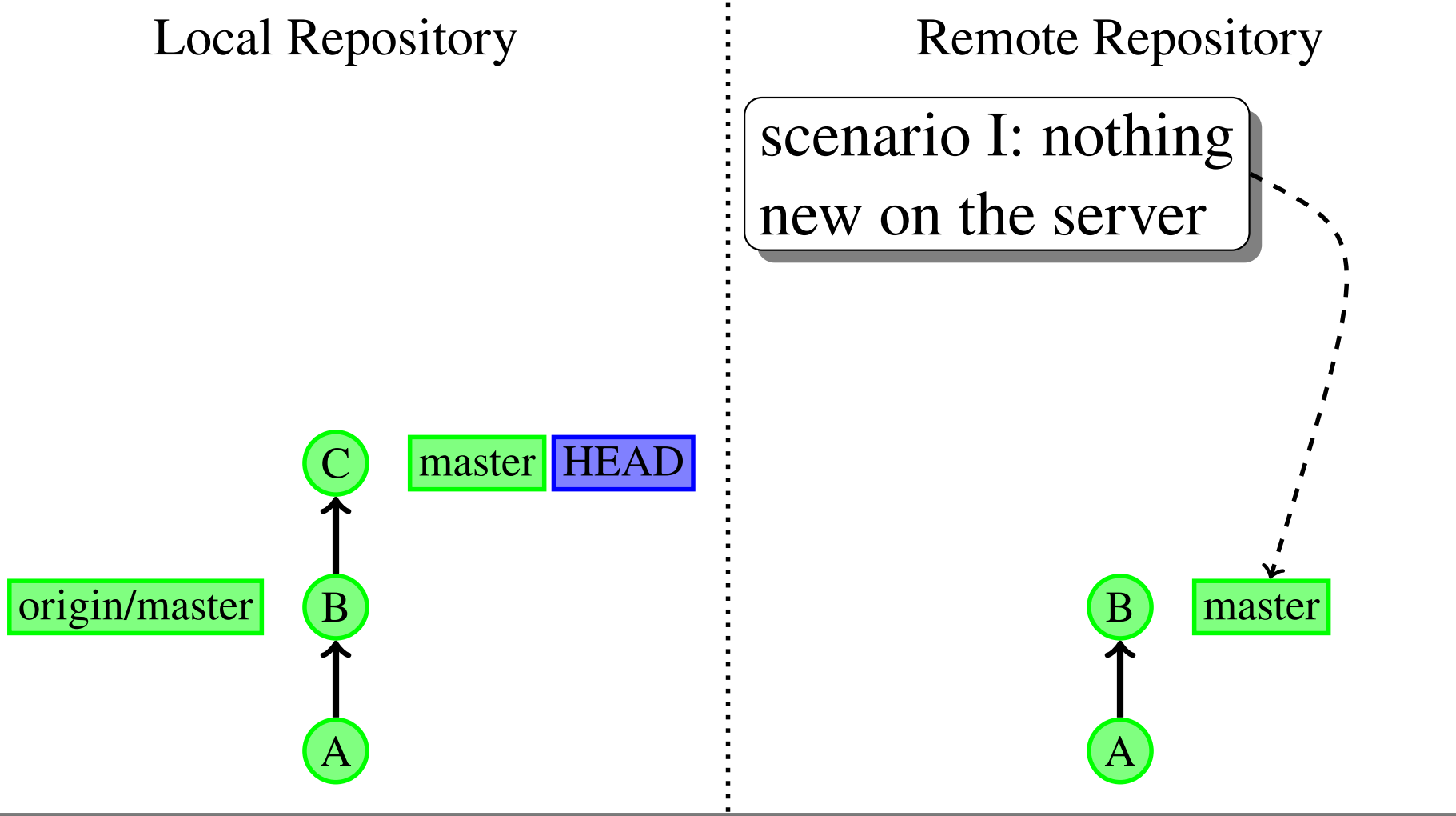
`revision1`	the first revision to compare, defaults to working copy
`revision2`	the second revision to compare, defaults to HEAD
`path`	shell pattern for file(s) to diff
`--cached`	use the staged content as `revision1`

or

A A

```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
git@test: git add newfile.txt
git@test: git commit
```

Uploading your changes I - no conflicts



Uploading your changes I - no conflicts

Local Repository

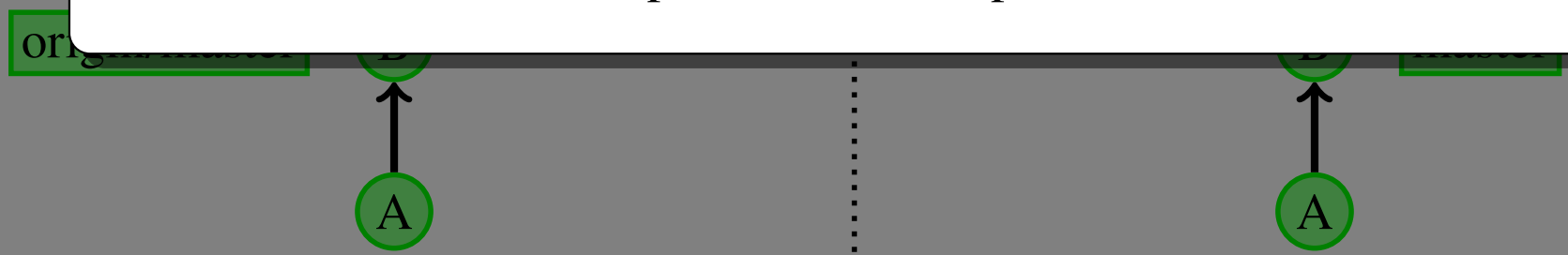
Remote Repository

`git fetch <remote> [<branch>]`

update all remote branches from the specified remote repository

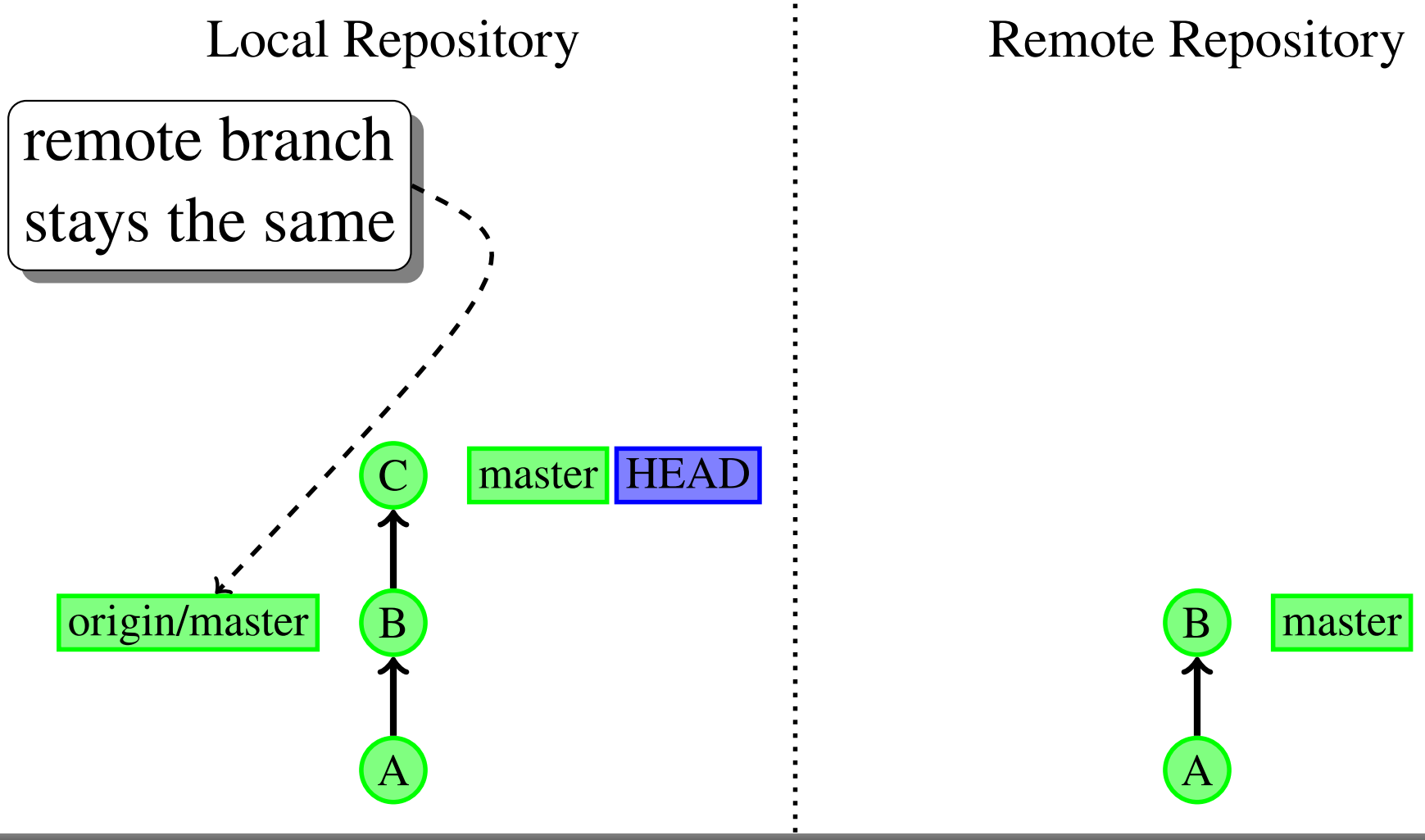
`remote` the remote repository to fetch from

`branch` limit the fetch operation to the specified branch




```
git@test: git fetch origin
```

Uploading your changes I - no conflicts



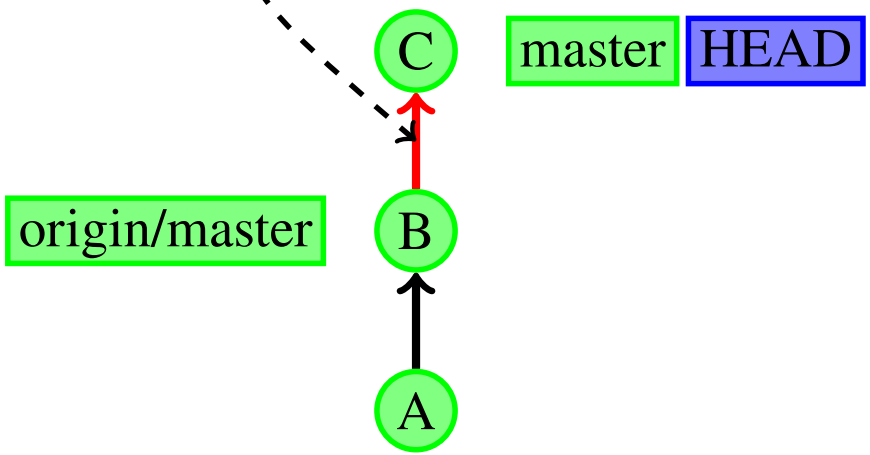

```
git@test: git fetch origin
```

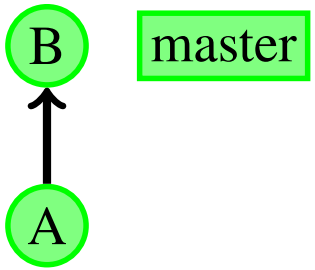
Uploading your changes I - no conflicts

Local Repository

Remote Repository

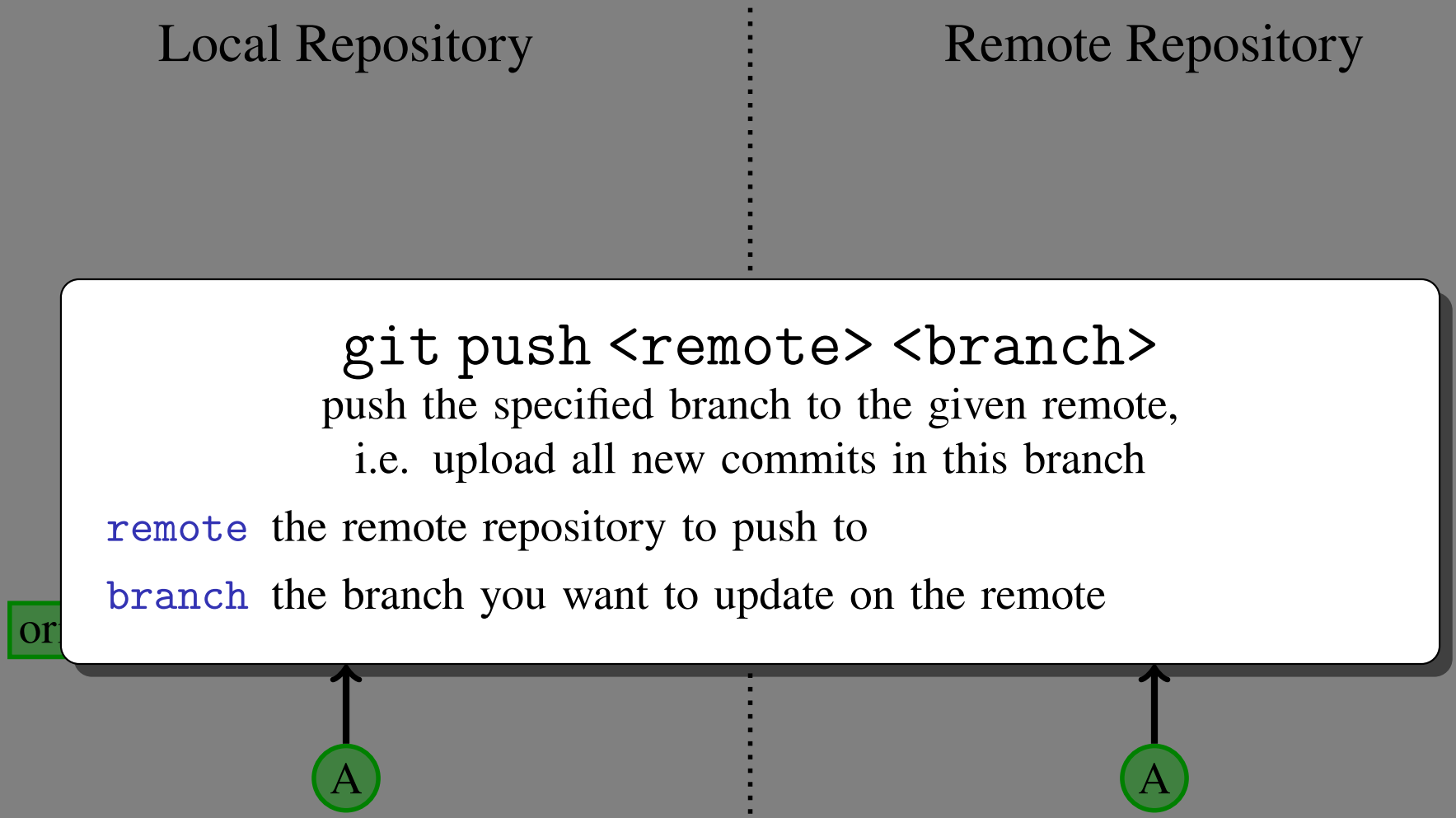
linear history connection
”fast forward” possible





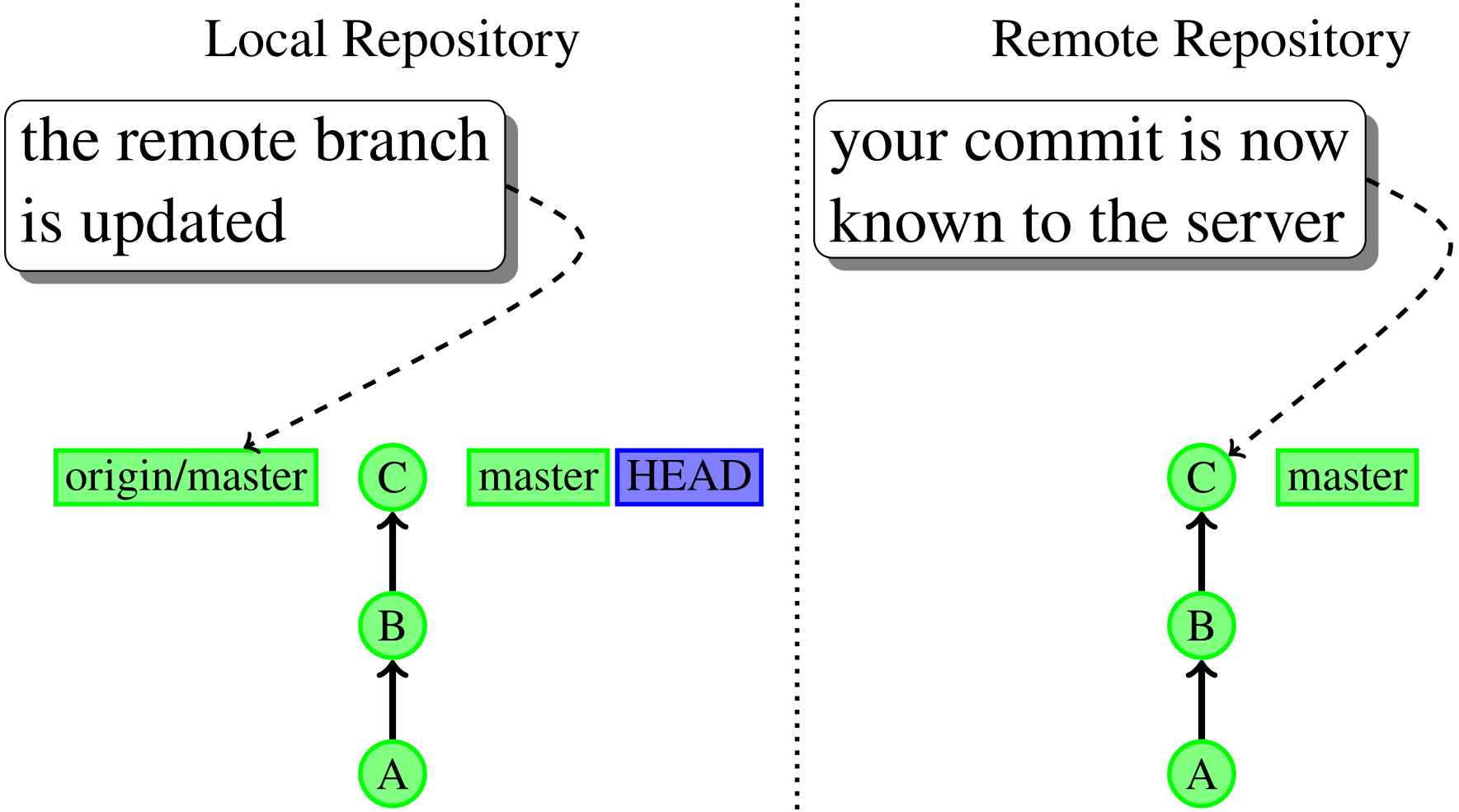

```
git@test: git fetch origin
```

Uploading your changes I - no conflicts



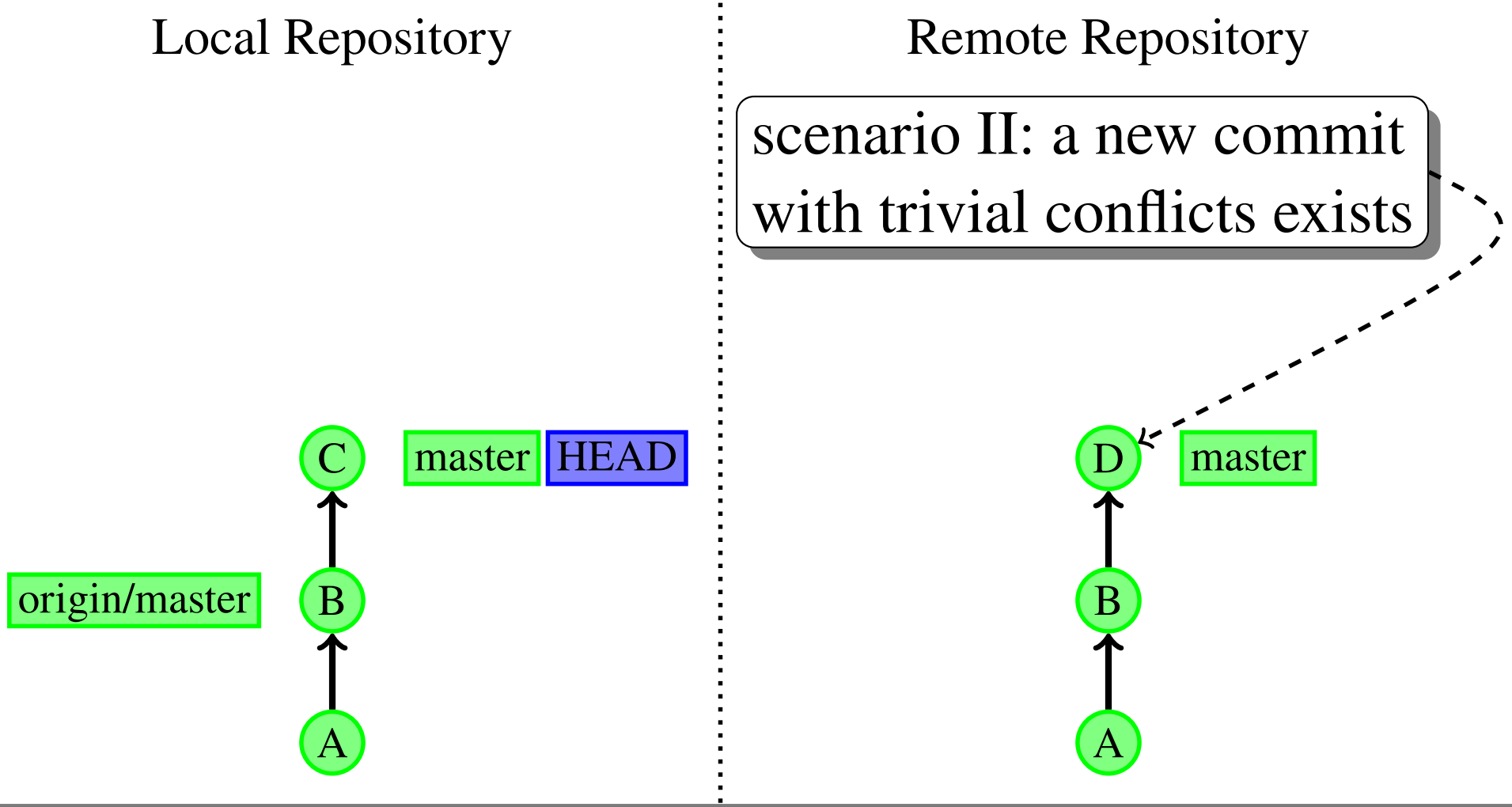

```
git@test: git fetch origin
git@test: git push origin master
```

Uploading your changes I - no conflicts



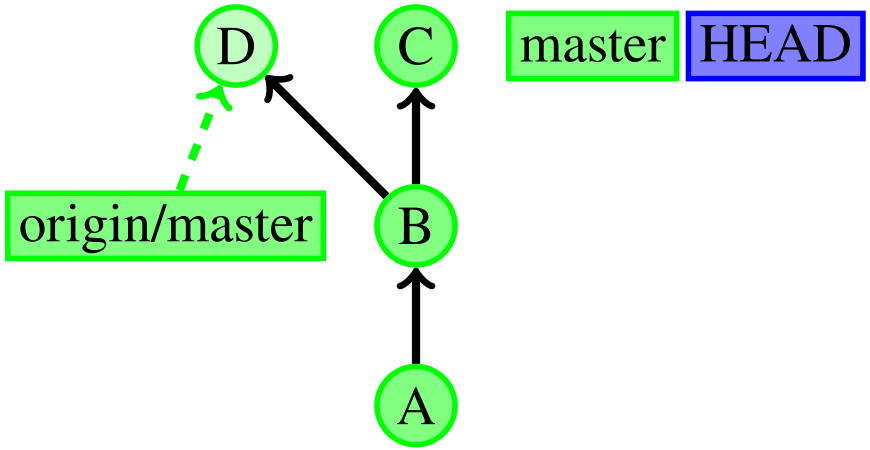

```
git@test: git fetch origin
git@test: git push origin master
```

Uploading your changes II - trivial conflicts

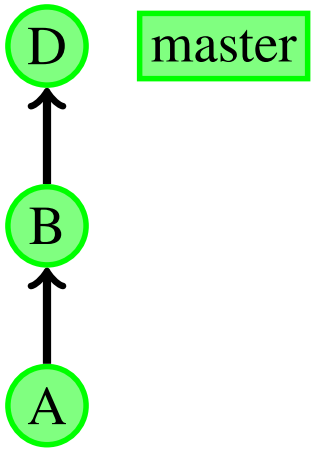


Uploading your changes II - trivial conflicts

Local Repository

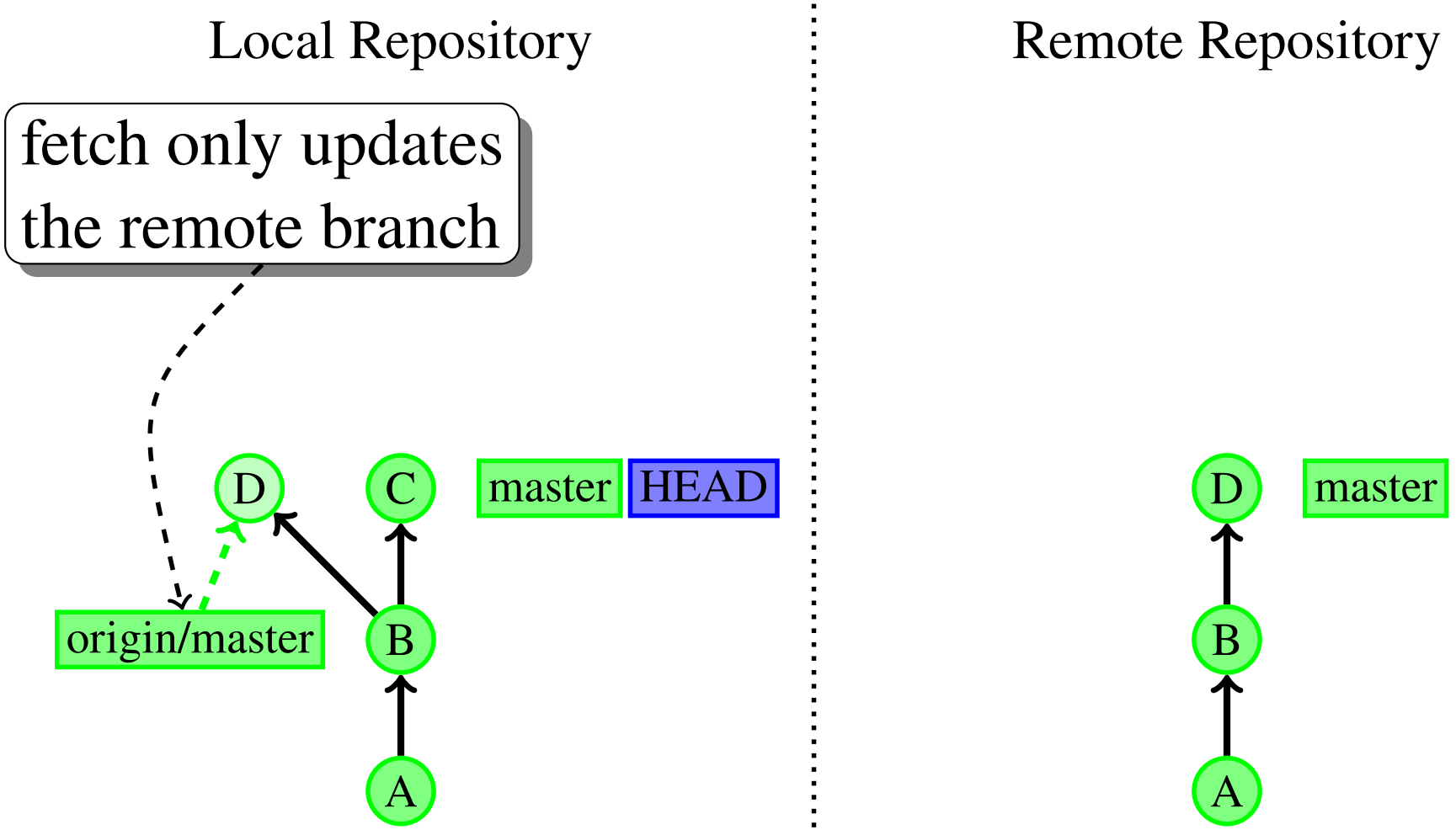


Remote Repository




```
git@test: git fetch origin
```

Uploading your changes II - trivial conflicts



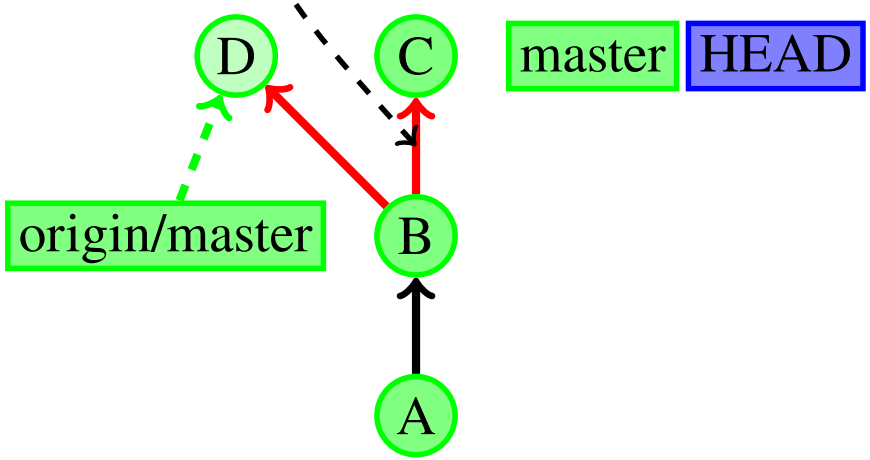

```
git@test: git fetch origin
```

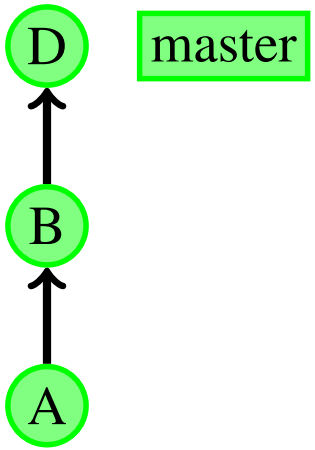
Uploading your changes II - trivial conflicts

Local Repository

Remote Repository

this fork in the history must be resolved






```
git@test: git fetch origin
```

Uploading your changes II - trivial conflicts

Local Repository | Remote Repository

`git rebase <branch>`

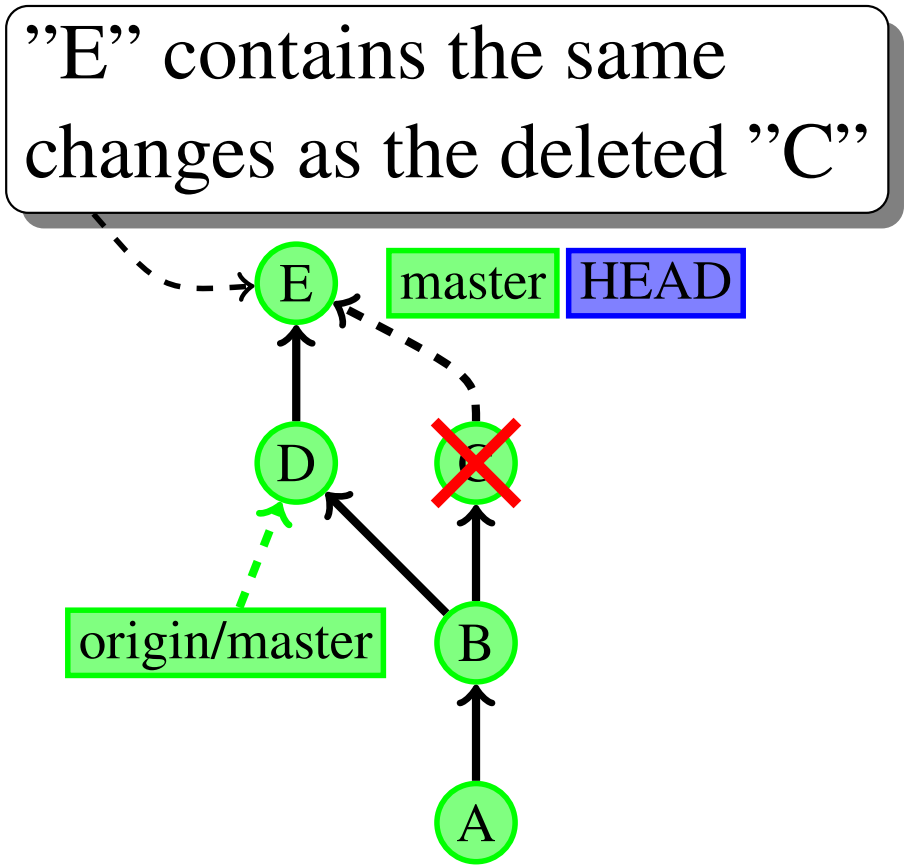
rewrite your new commits in the current branch
to be based on another parent commit

`branch` the branch whose tip to use as the new ancestor commit

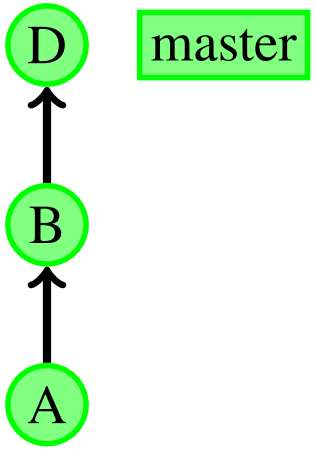

```
git@test: git fetch origin
git@test: git rebase origin/master
```

Uploading your changes II - trivial conflicts

Local Repository



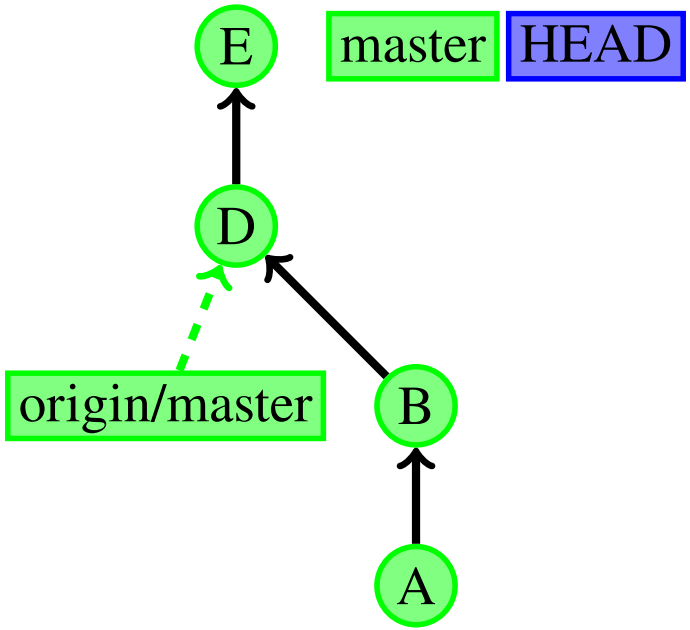
Remote Repository



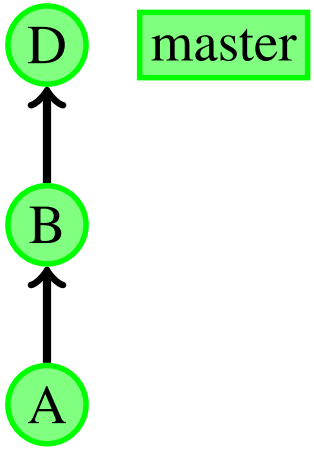

```
git@test: git fetch origin
git@test: git rebase origin/master
```

Uploading your changes II - trivial conflicts

Local Repository



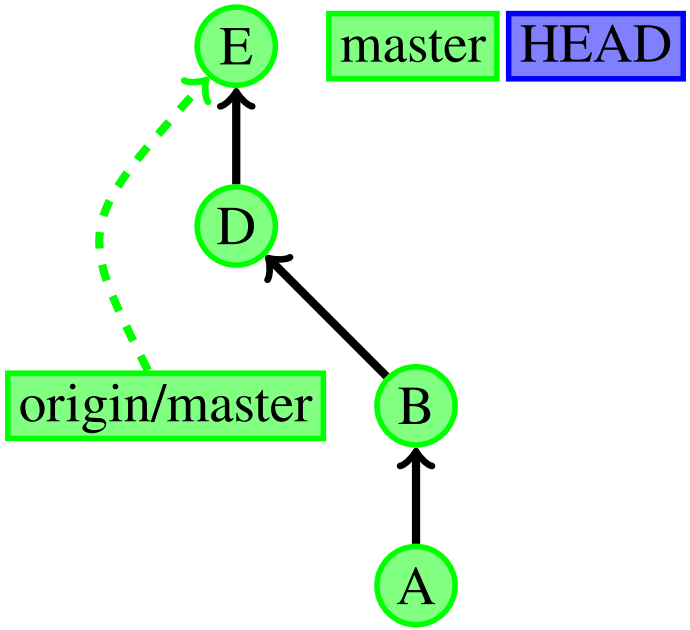
Remote Repository



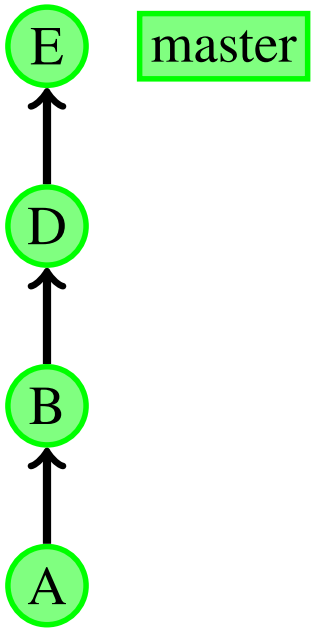

```
git@test: git fetch origin
git@test: git rebase origin/master
```

Uploading your changes II - trivial conflicts

Local Repository



Remote Repository



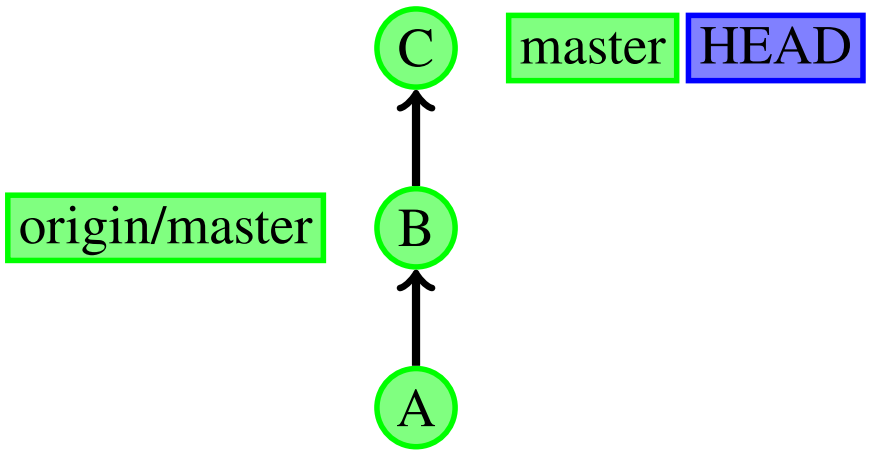

```
git@test: git fetch origin
git@test: git rebase origin/master
git@test: git push origin master
```

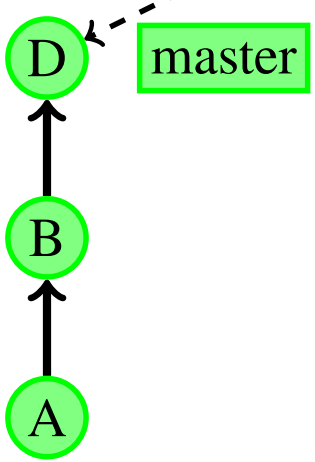
Uploading your changes III - real conflicts

Local Repository

Remote Repository

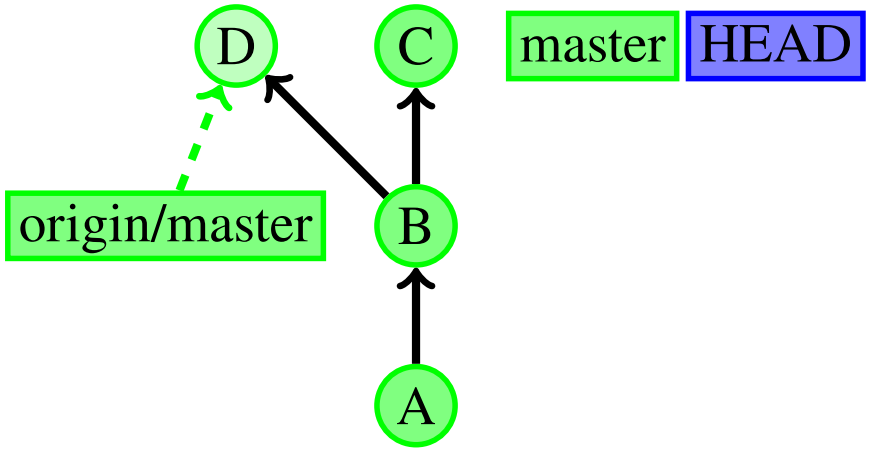
scenario III: a new commit with real conflicts exists



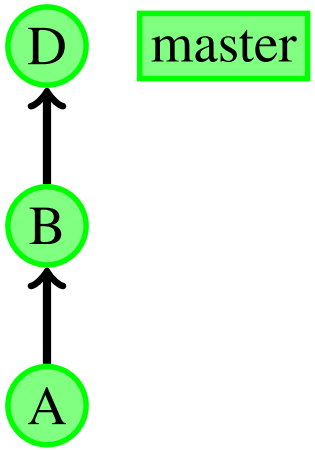


Uploading your changes III - real conflicts

Local Repository



Remote Repository



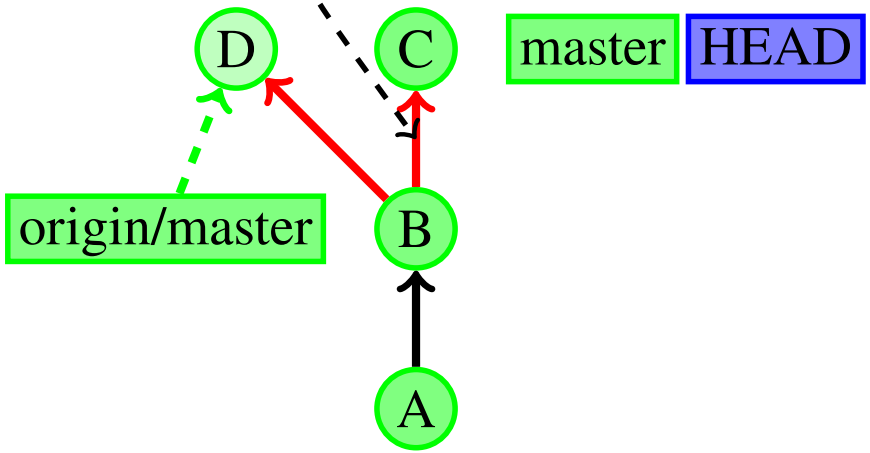

```
git@test: git fetch origin
```

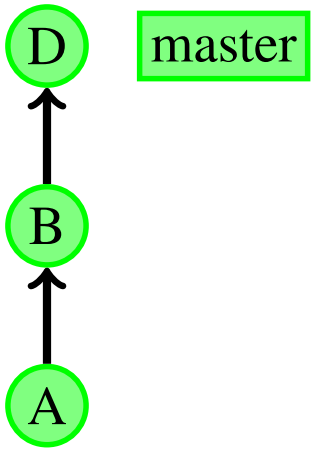
Uploading your changes III - real conflicts

Local Repository

Remote Repository

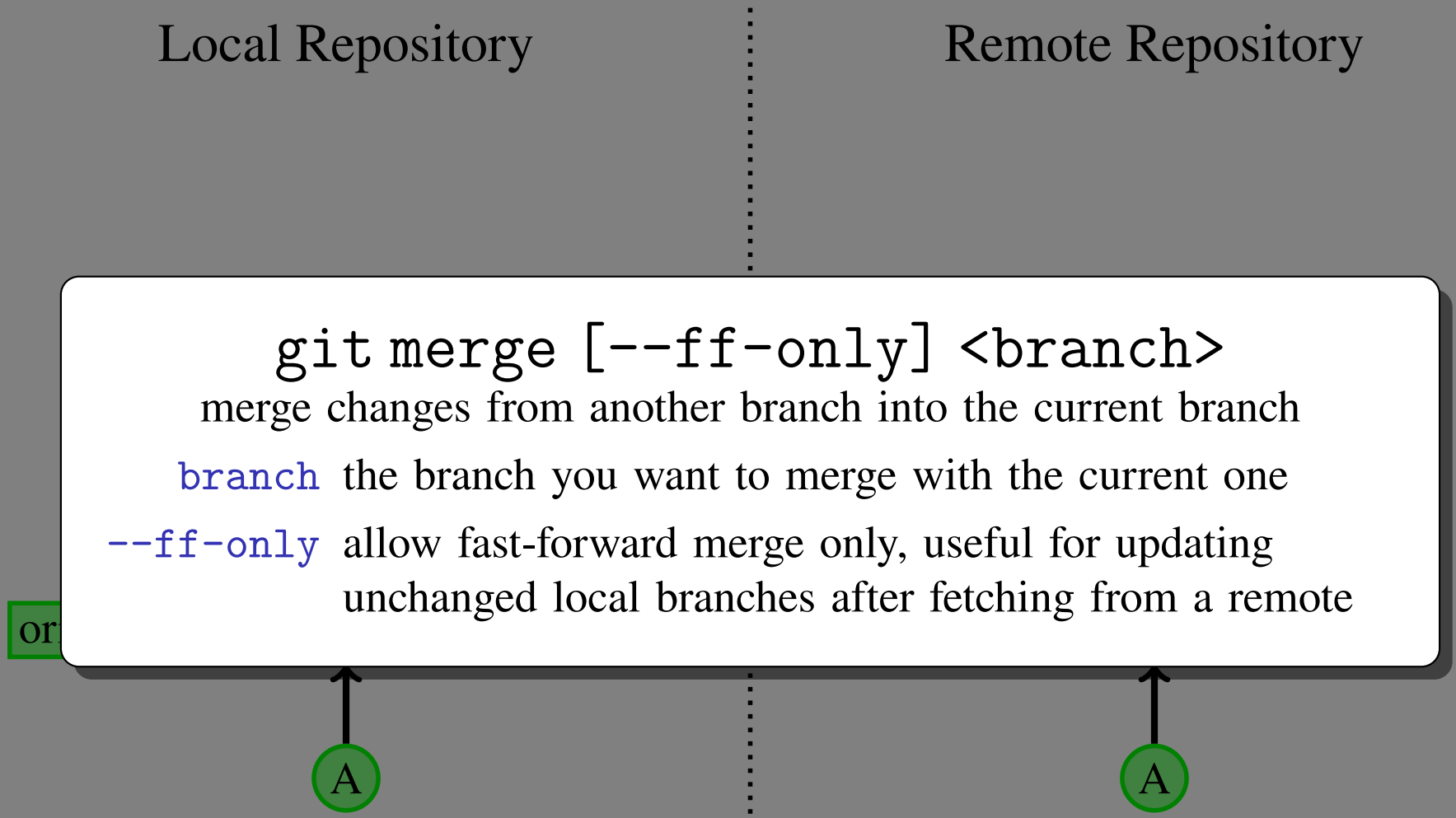
conflicting changes must be resolved






```
git@test: git fetch origin
```

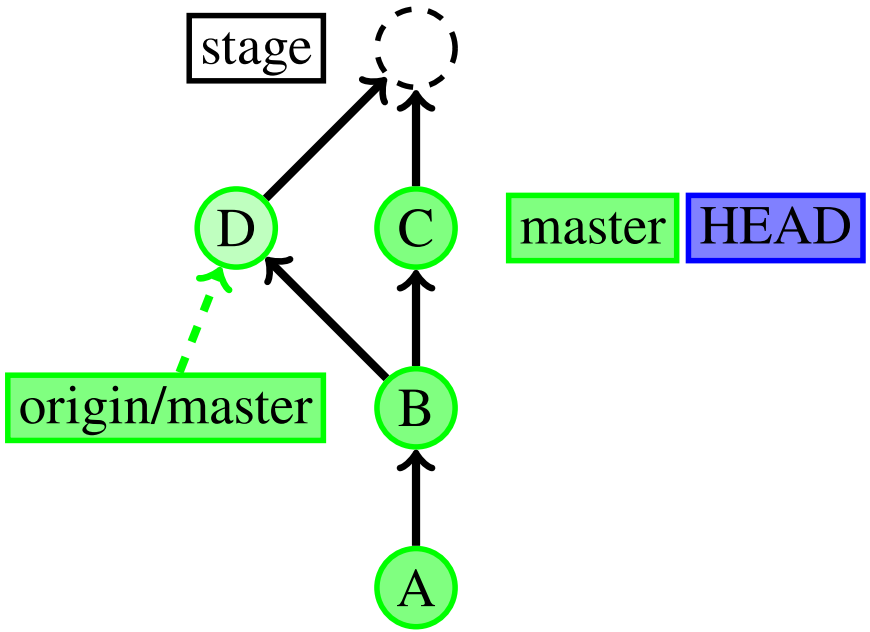
Uploading your changes III - real conflicts



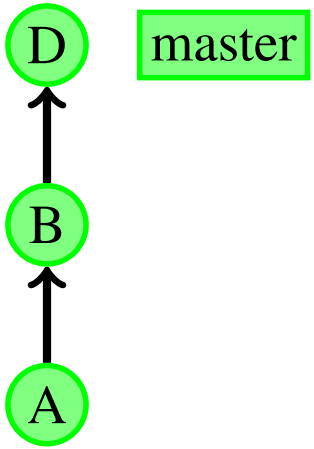

```
git@test: git fetch origin
git@test: git merge origin/master
```

Uploading your changes III - real conflicts

Local Repository



Remote Repository

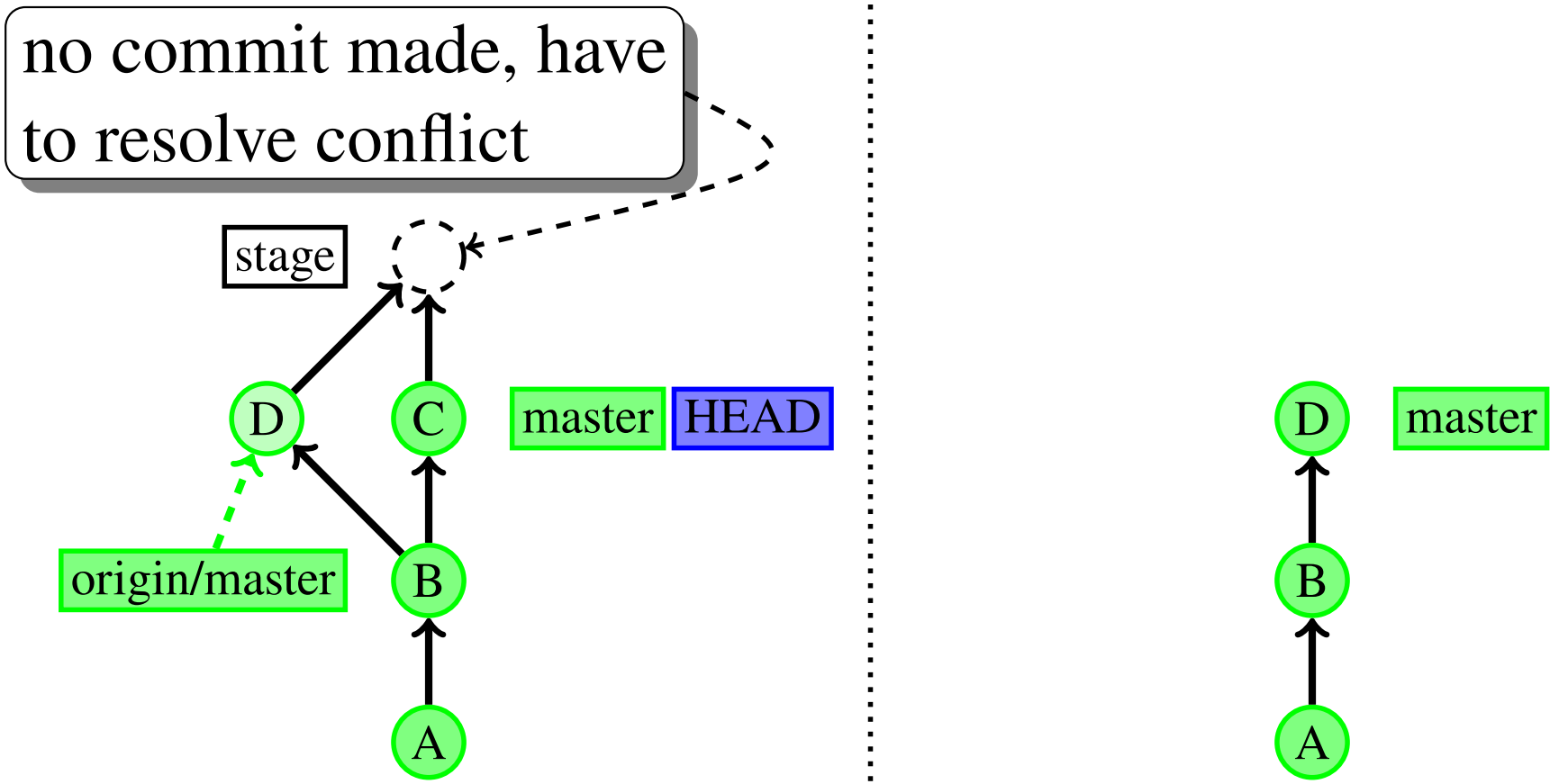



```
git@test: git fetch origin
git@test: git merge origin/master
```

Uploading your changes III - real conflicts

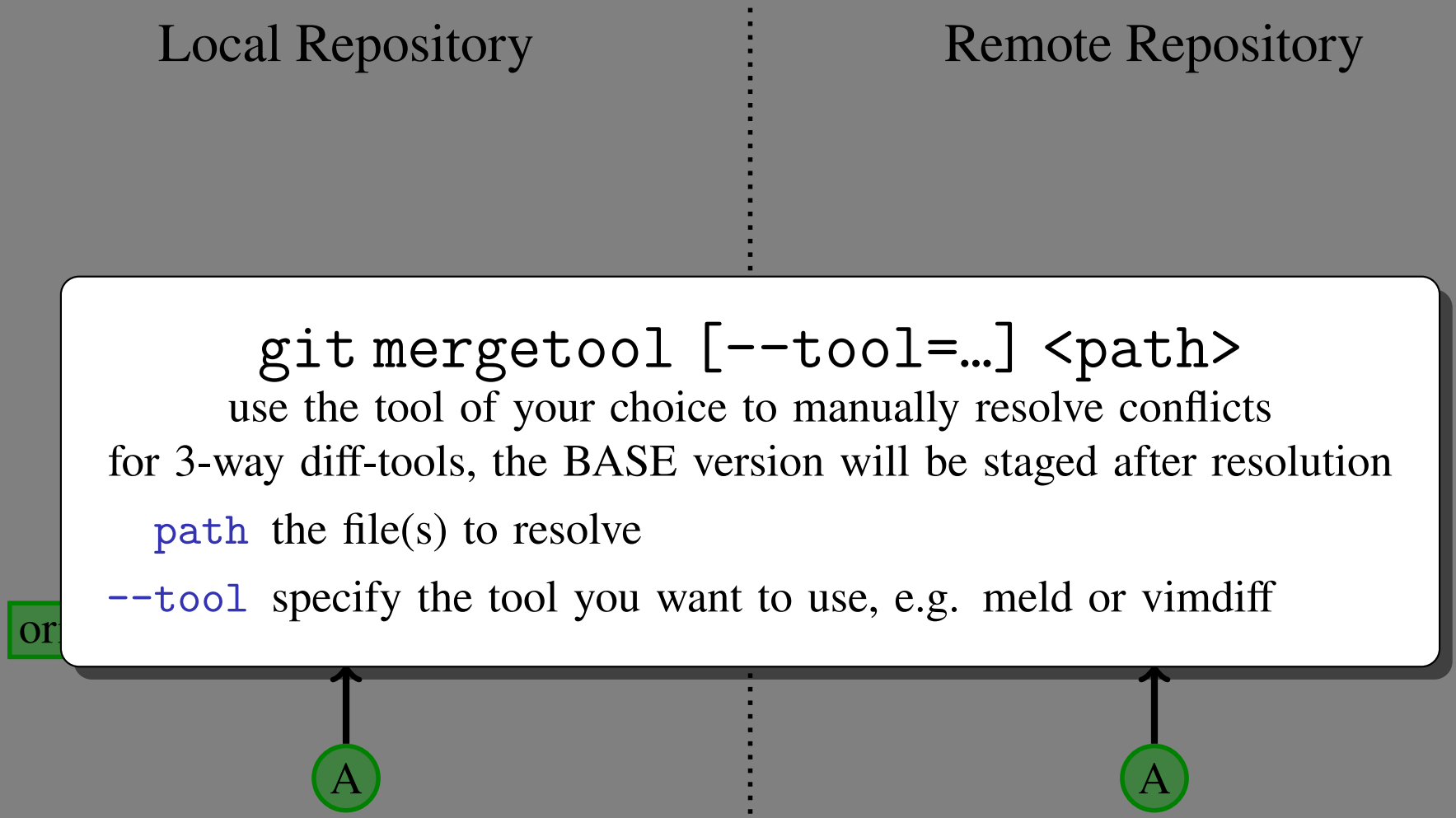
Local Repository

Remote Repository




```
git@test: git fetch origin
git@test: git merge origin/master
```

Uploading your changes III - real conflicts

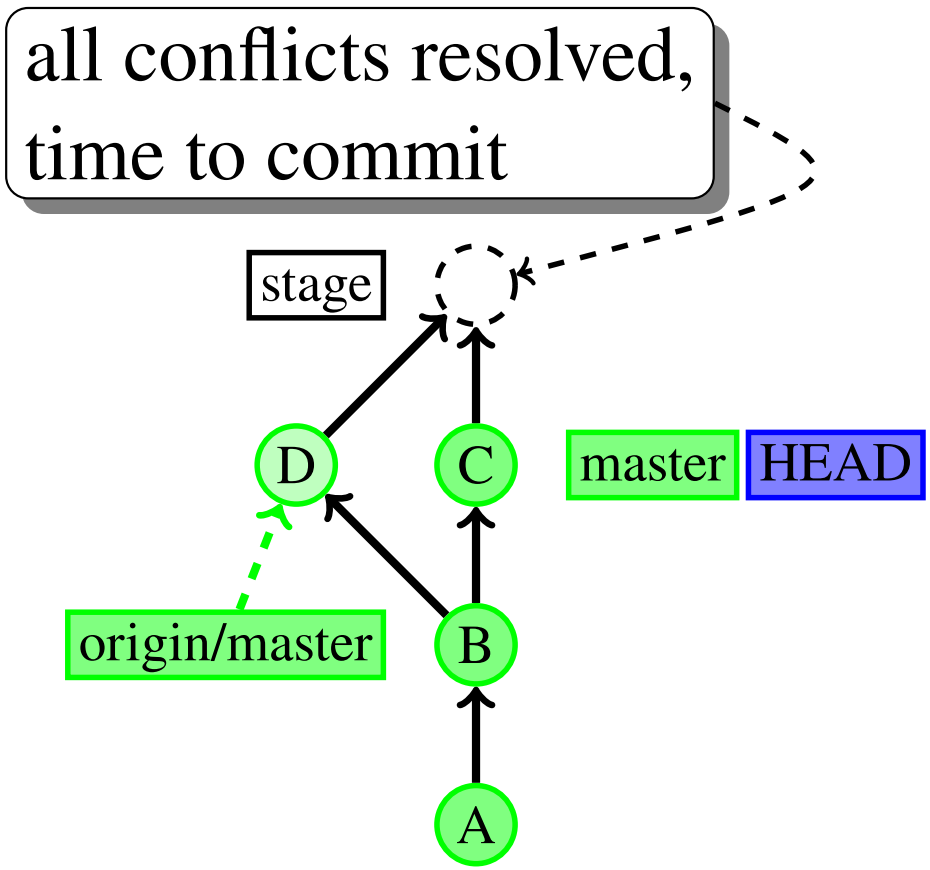



```
git@test: git fetch origin
git@test: git merge origin/master
git@test: git mergetool nasty_conflict.py
```

Uploading your changes III - real conflicts

Local Repository

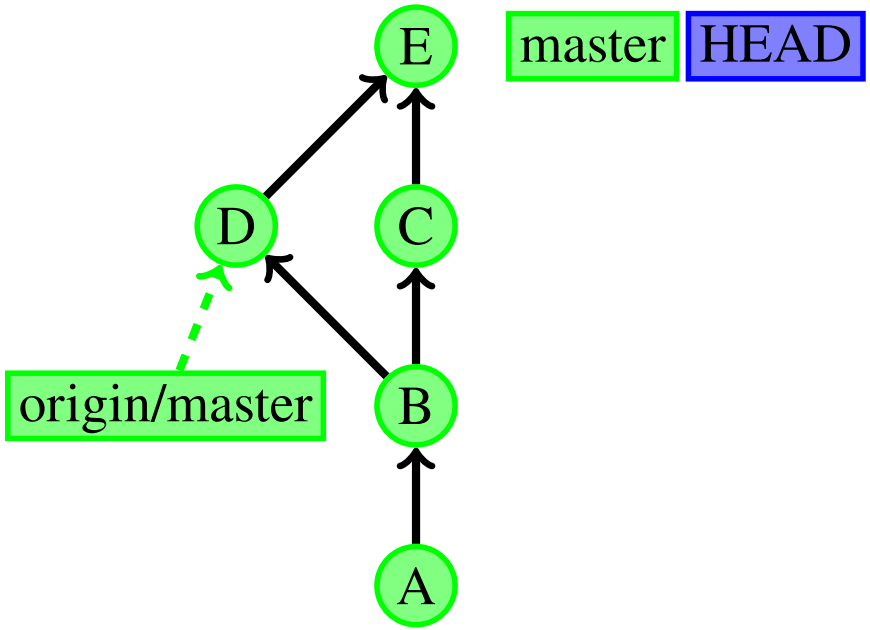
Remote Repository




```
git@test: git fetch origin
git@test: git merge origin/master
git@test: git mergetool nasty_conflict.py
```

Uploading your changes III - real conflicts

Local Repository

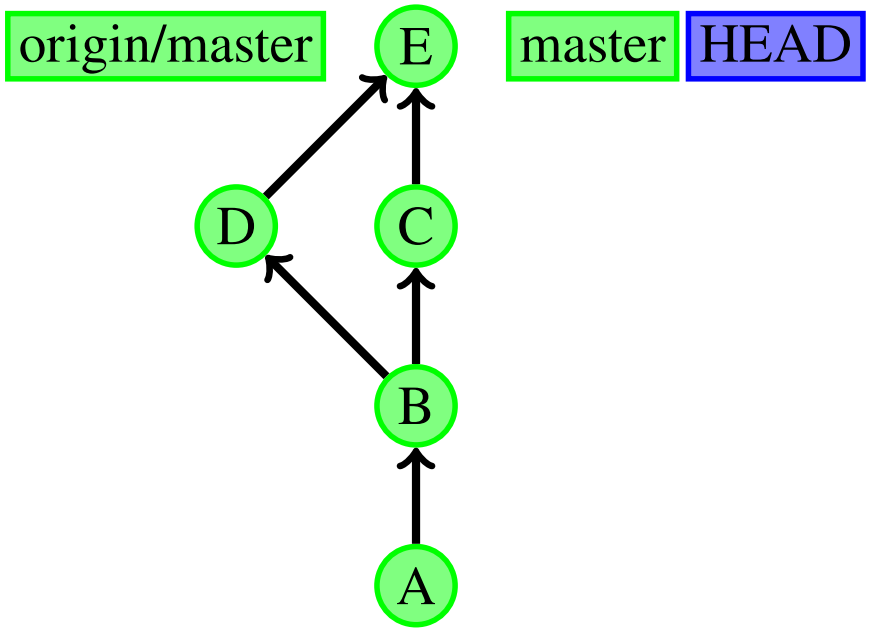


Remote Repository

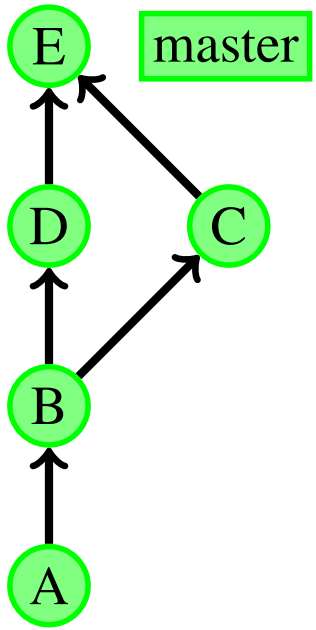

```
git@test: git fetch origin
git@test: git merge origin/master
git@test: git mergetool nasty_conflict.py
git@test: git commit
```

Uploading your changes III - real conflicts

Local Repository



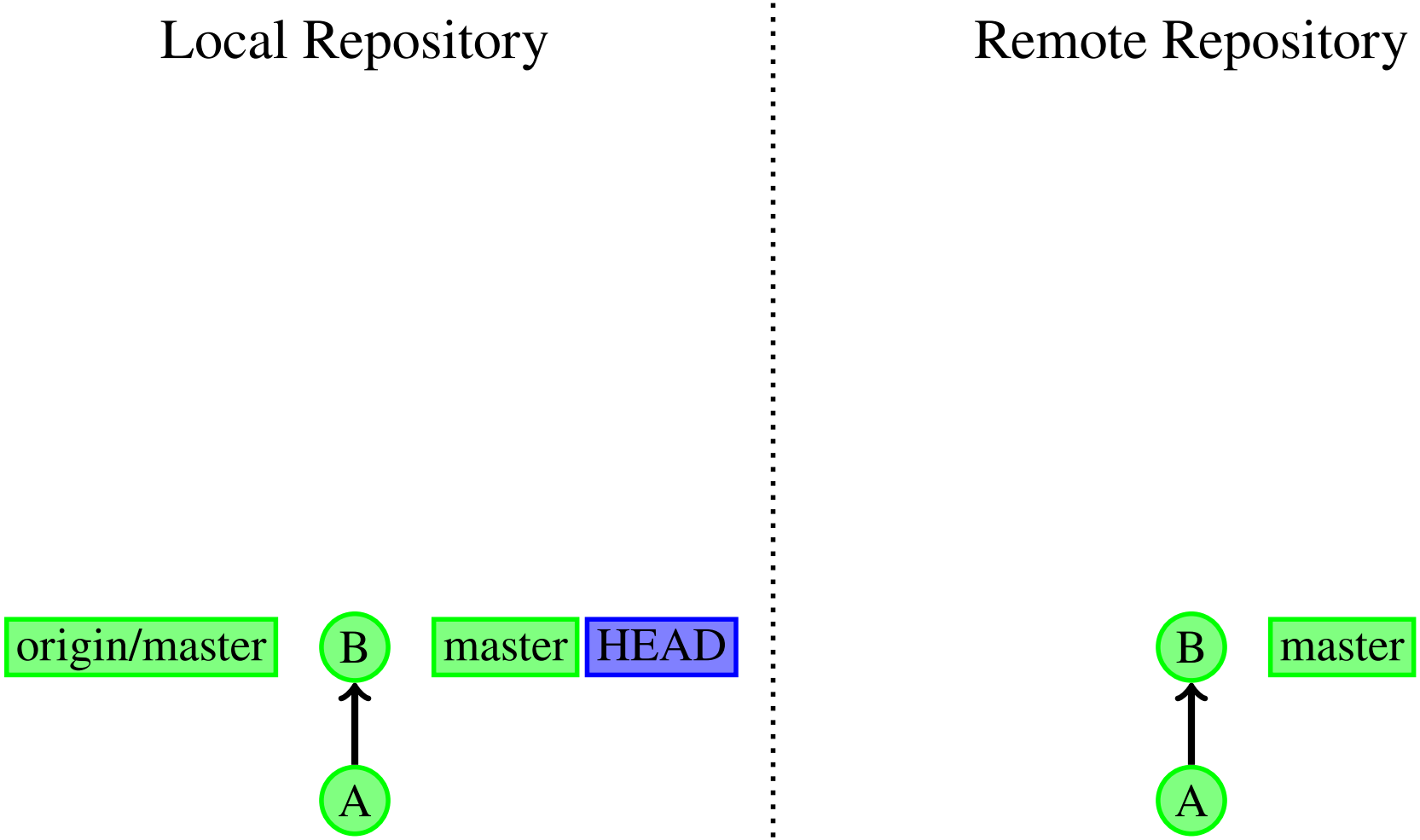
Remote Repository




```
git@test: git fetch origin
git@test: git merge origin/master
git@test: git mergetool nasty_conflict.py
git@test: git commit
git@test: git push origin master
```

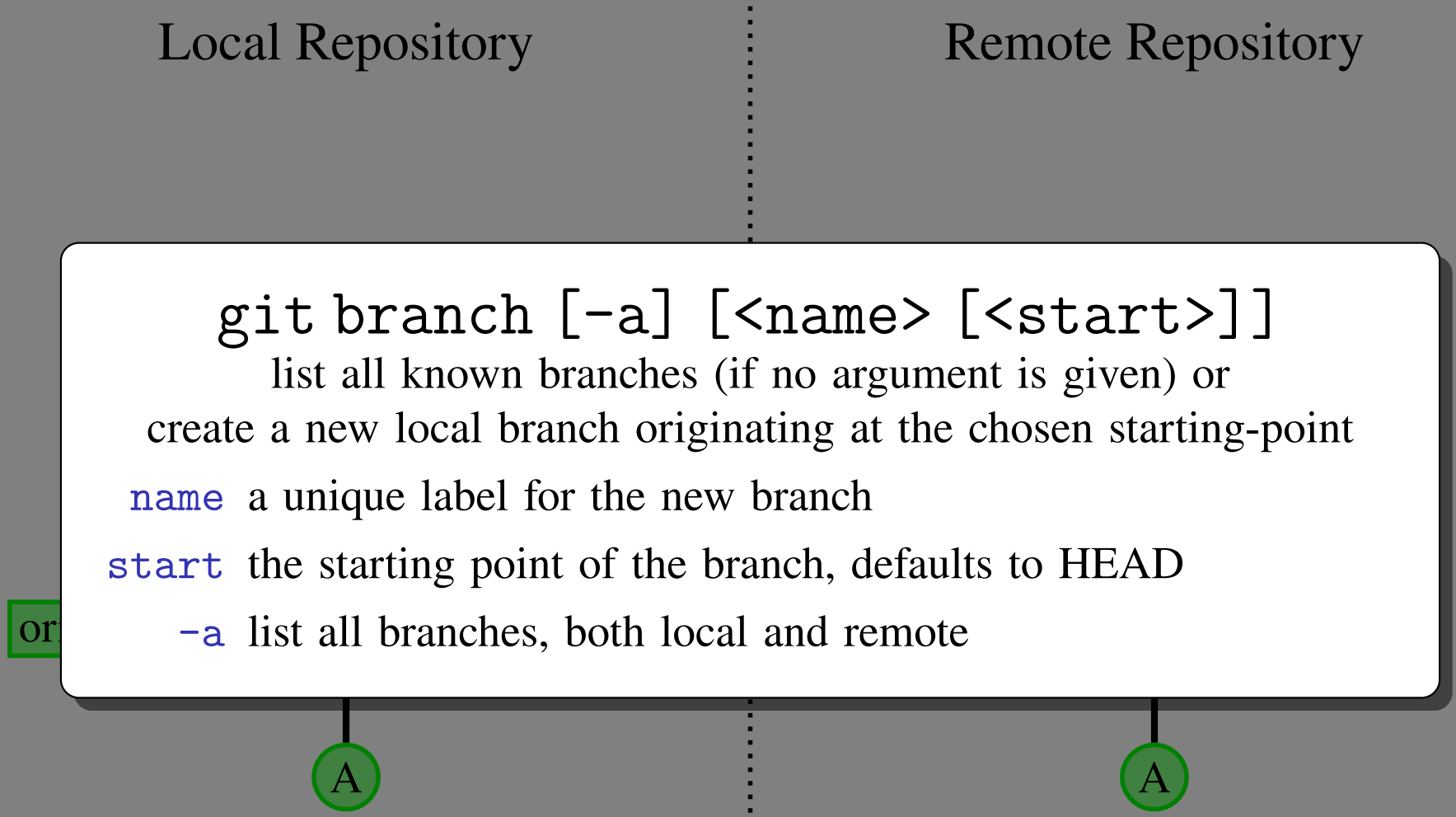
git branches
organizing and isolating your development

Committing your changes into a new local branch



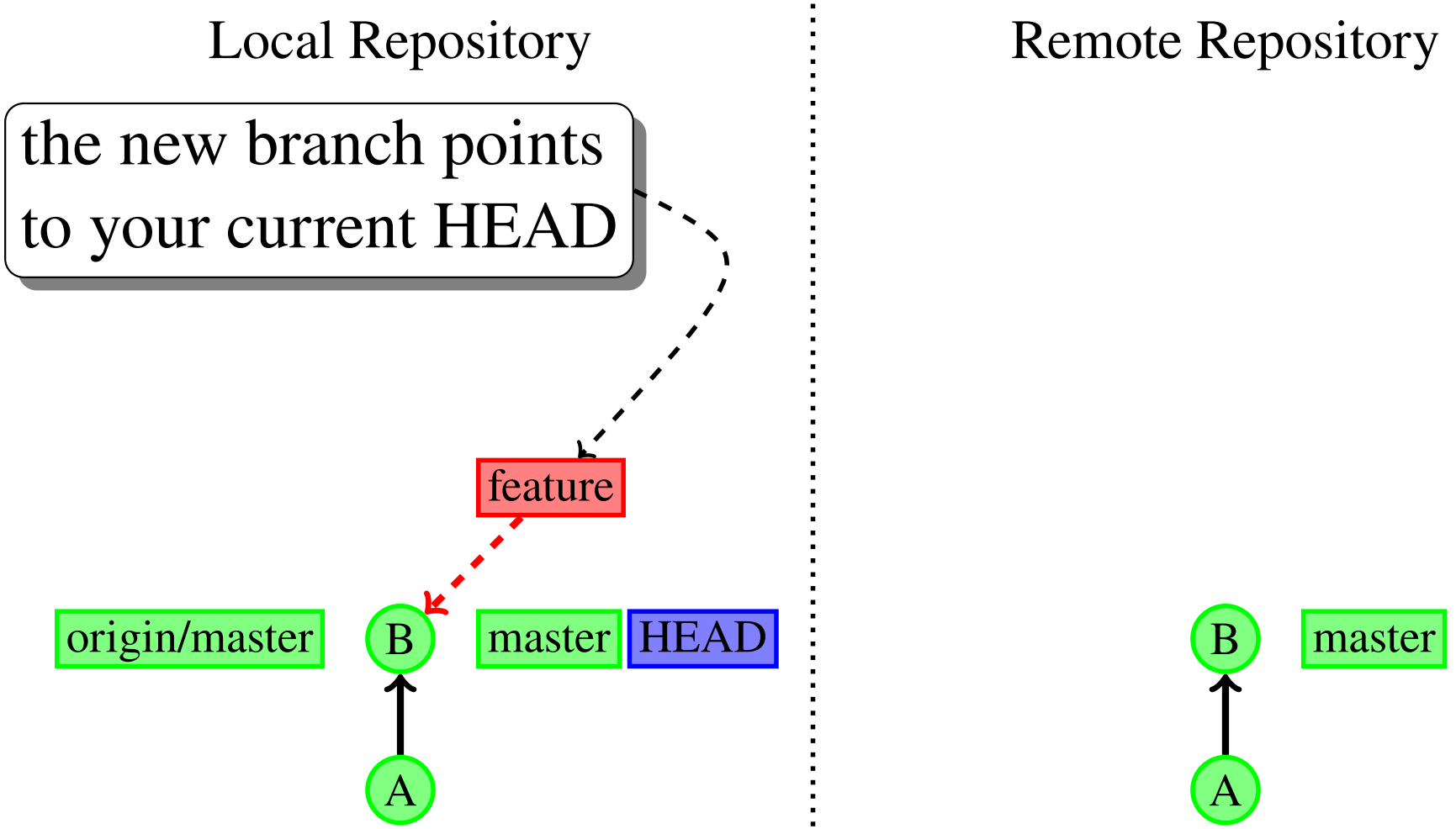

```
git@test: echo "This should be fun!" > newfile.txt
```

Committing your changes into a new local branch




```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
```

Committing your changes into a new local branch




```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
```

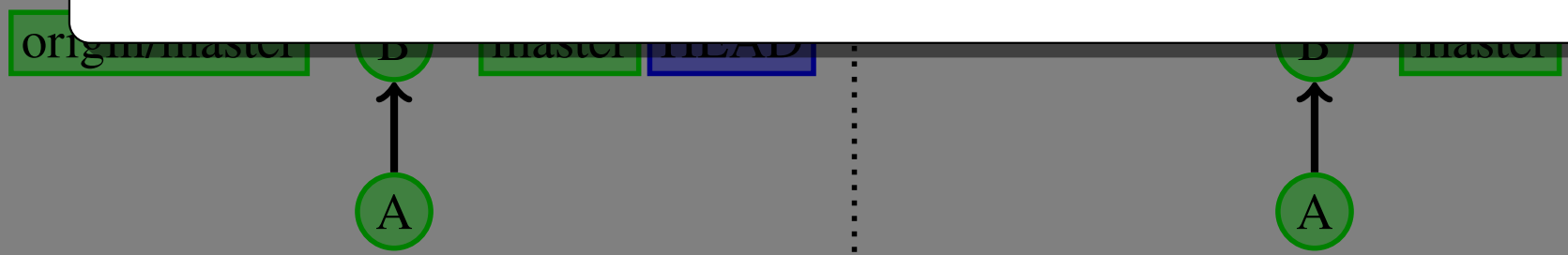
Committing your changes into a new local branch

Local Repository | Remote Repository

`git checkout <revision>`

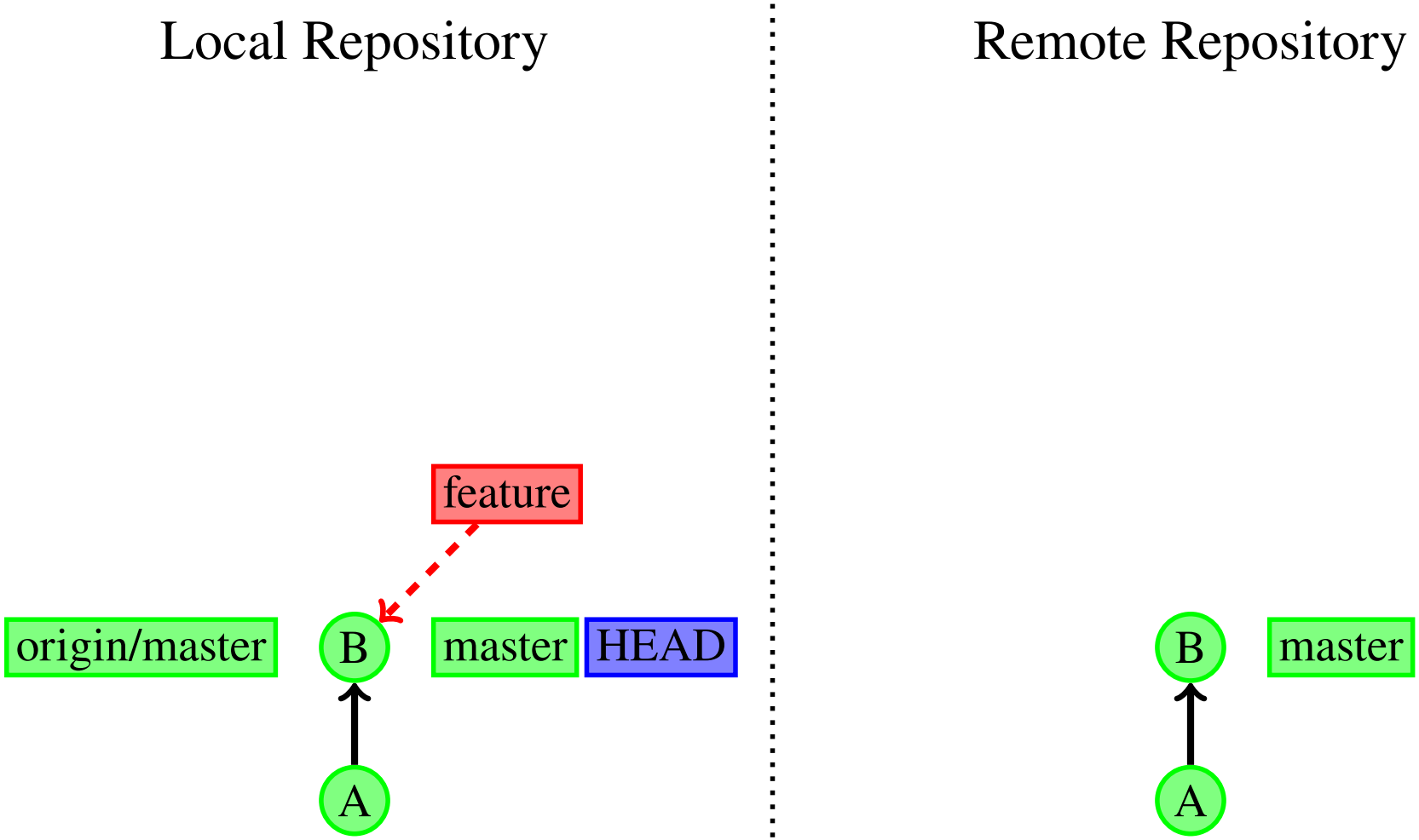
changes the working directory to the specified revision

`revision` the name of the target revision, can be a branch name, tag or commit-ID



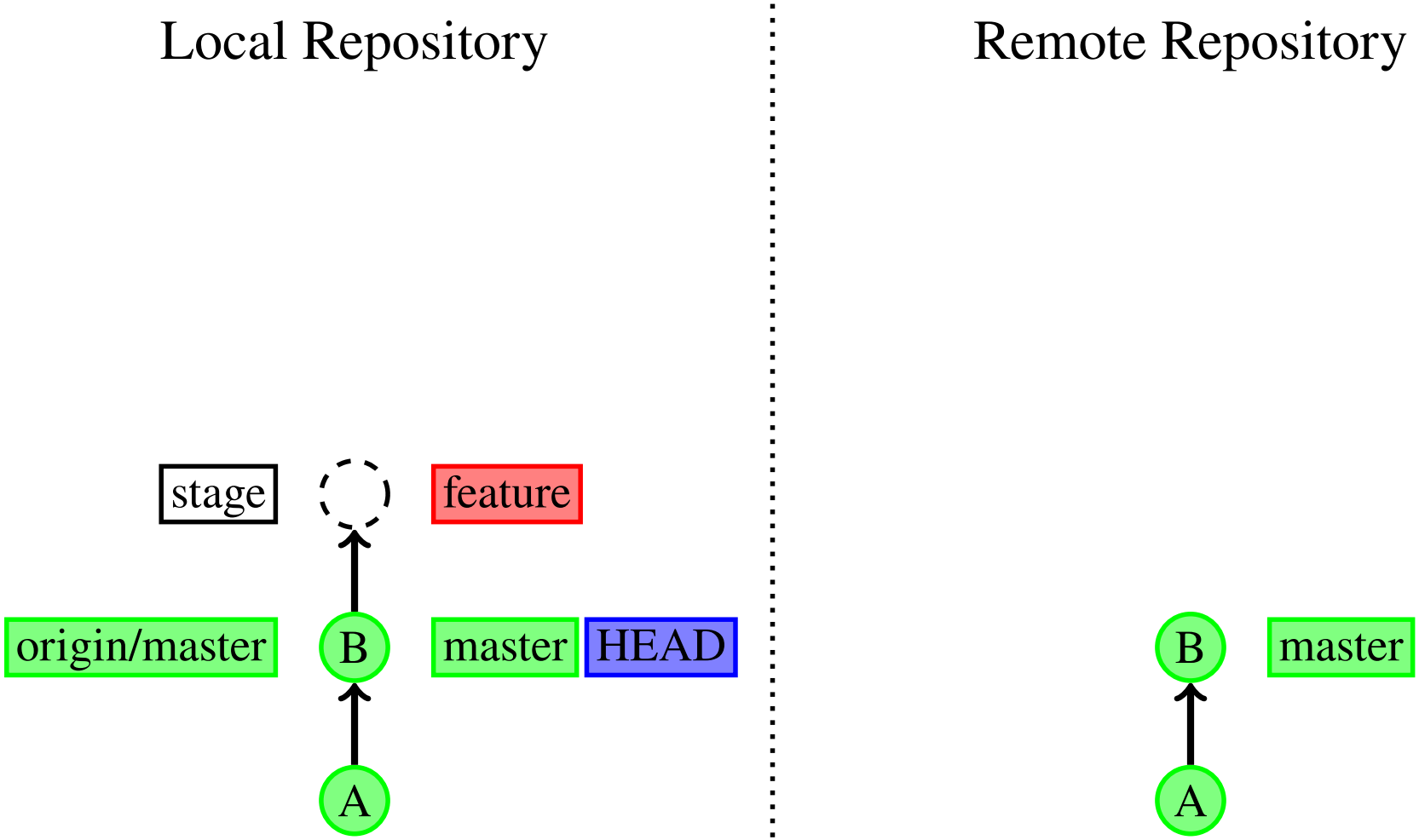

```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
```

Committing your changes into a new local branch




```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
```

Committing your changes into a new local branch



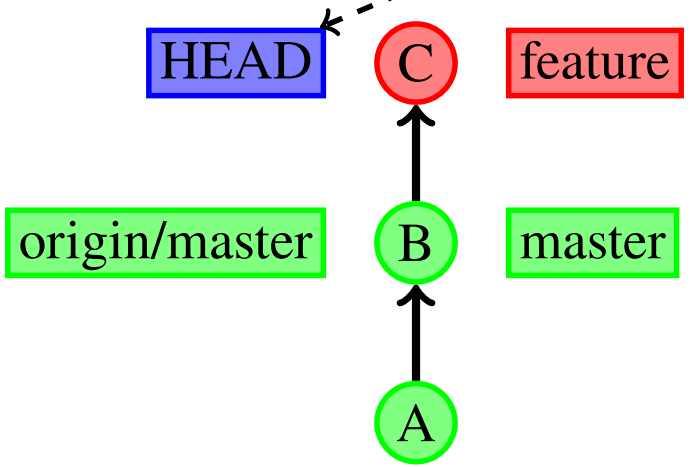

```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
git@test: git add newfile.txt
```

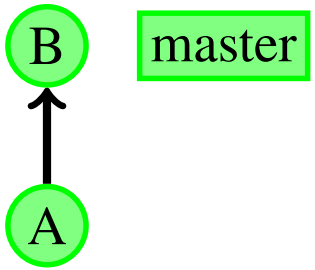
Committing your changes into a new local branch

Local Repository

Remote Repository

HEAD moved to the new working copy state





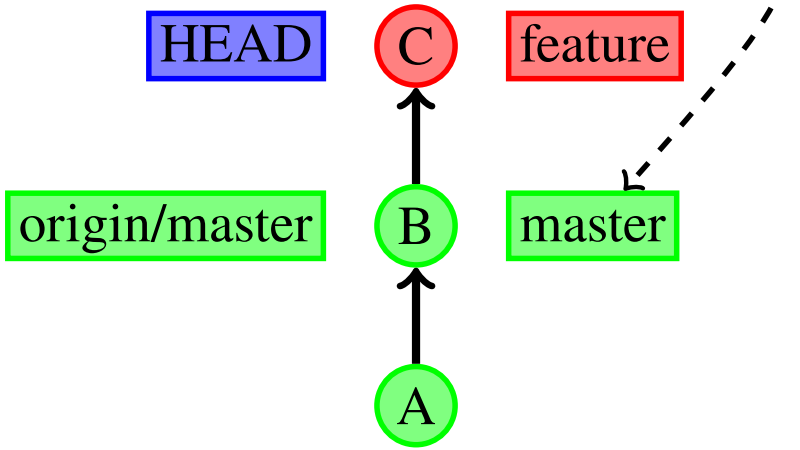

```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
git@test: git add newfile.txt
git@test: git commit
```

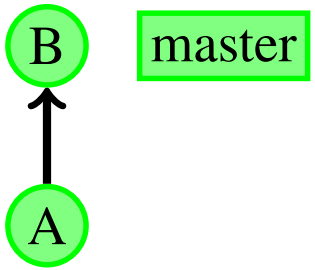
Committing your changes into a new local branch

Local Repository

Remote Repository

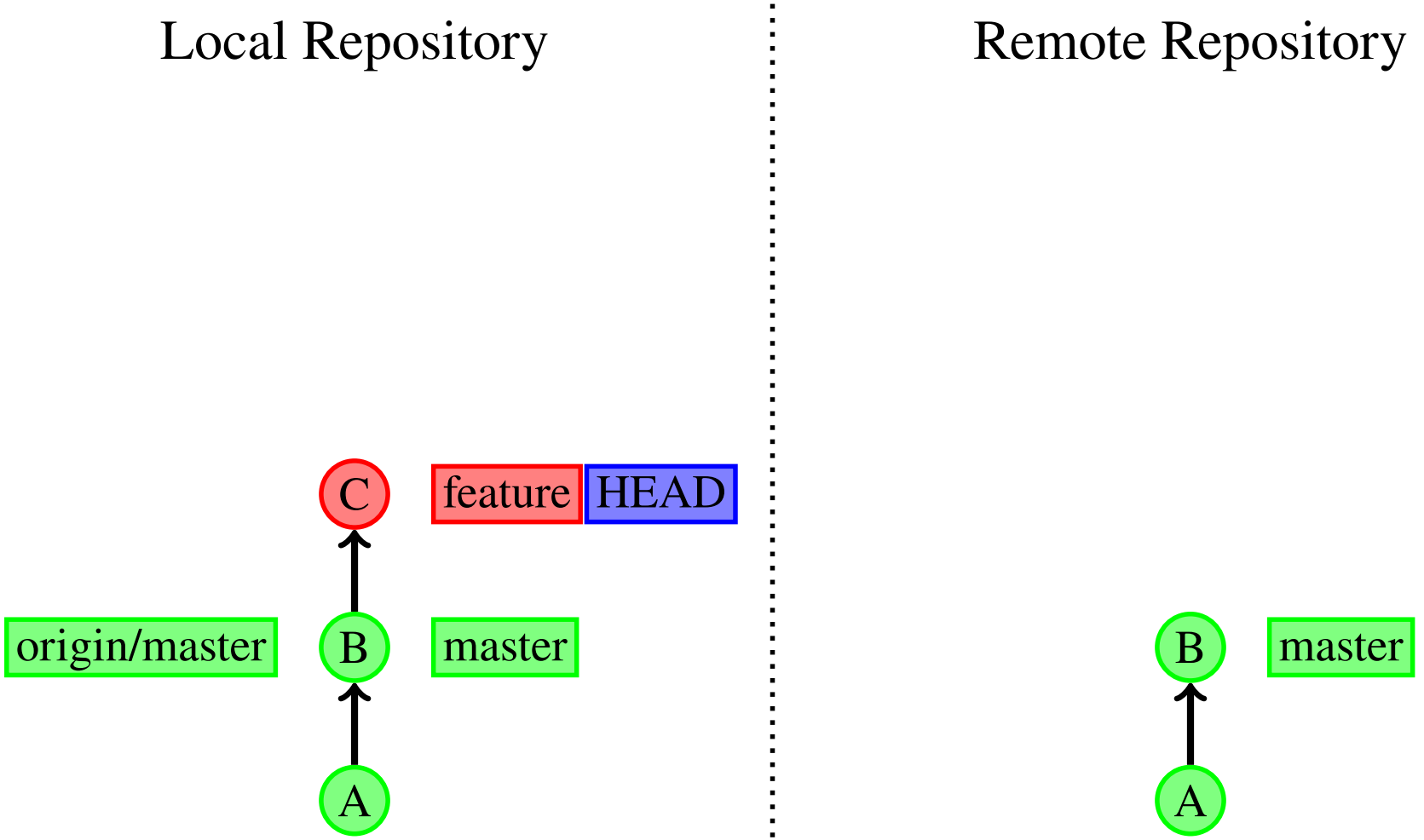
the master branch is unchanged






```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
git@test: git add newfile.txt
git@test: git commit
```

Uploading your changes IV - a new branch

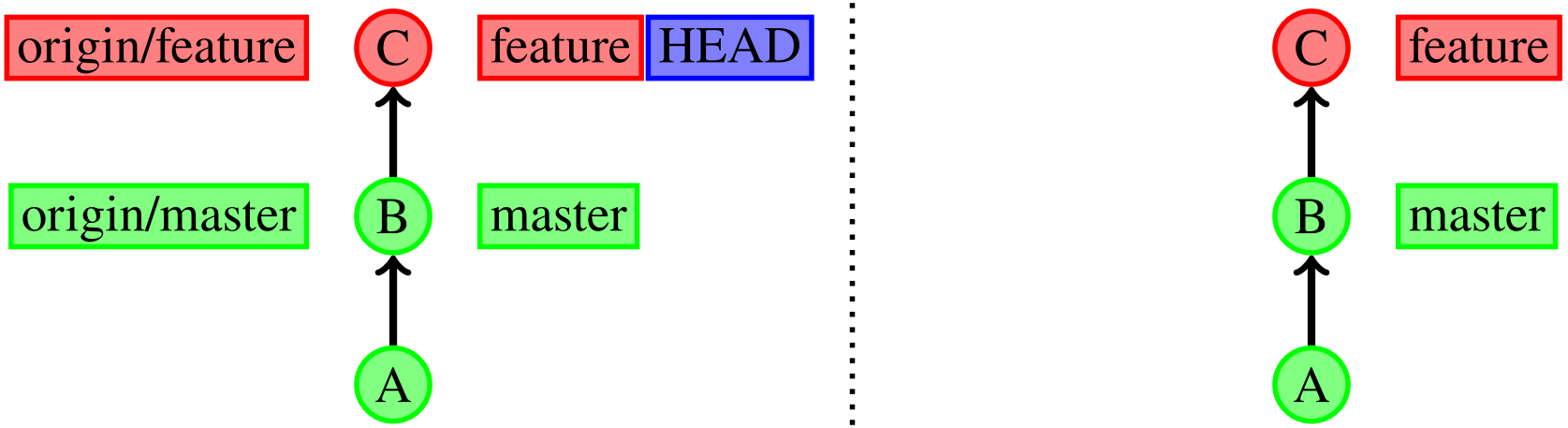



```
git@test: git fetch origin
```

Uploading your changes IV - a new branch

Local Repository

Remote Repository



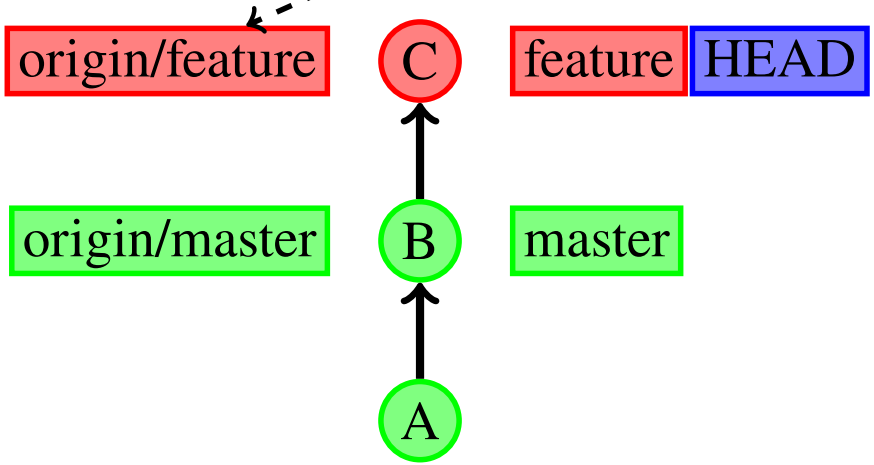

```
git@test: git fetch origin
git@test: git push origin feature
```

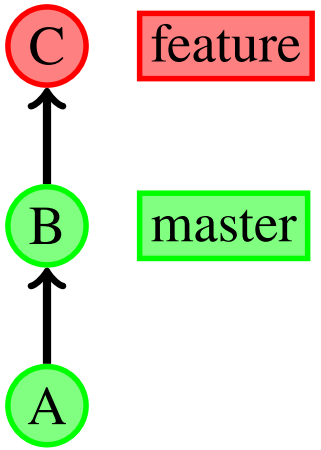
Uploading your changes IV - a new branch

Local Repository

Remote Repository

a new remote branch has been created

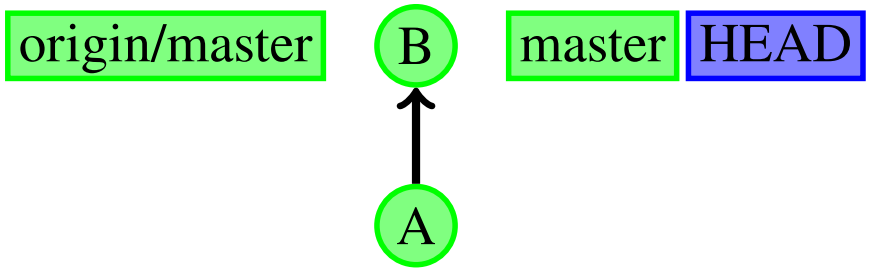




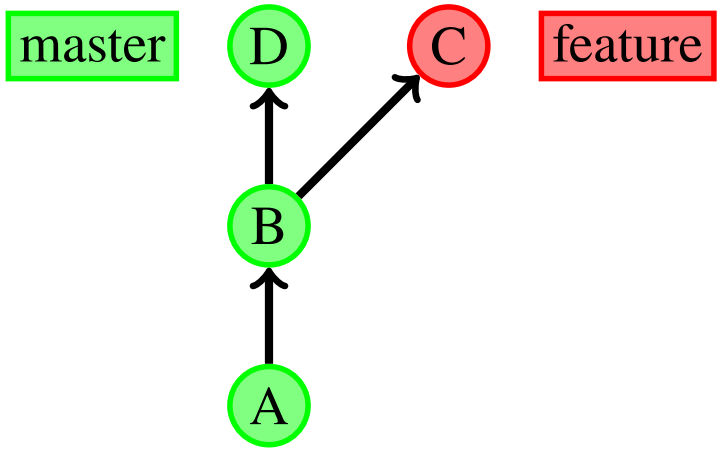

```
git@test: git fetch origin
git@test: git push origin feature
```

Working with branches - turning remote into local branches

Local Repository

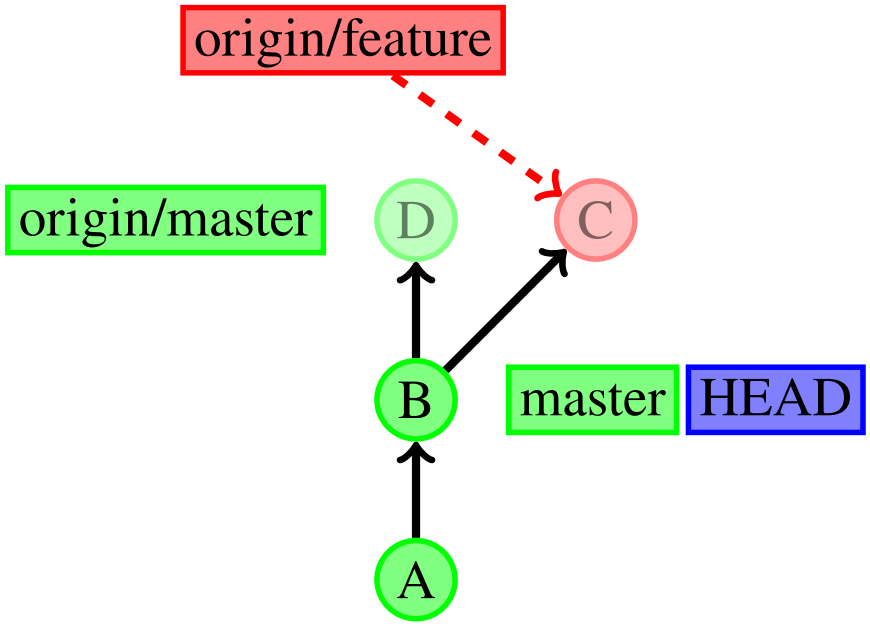


Remote Repository

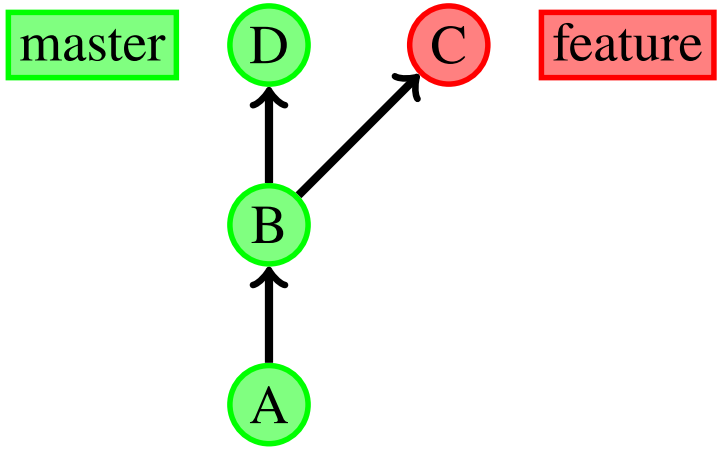


Working with branches - turning remote into local branches

Local Repository

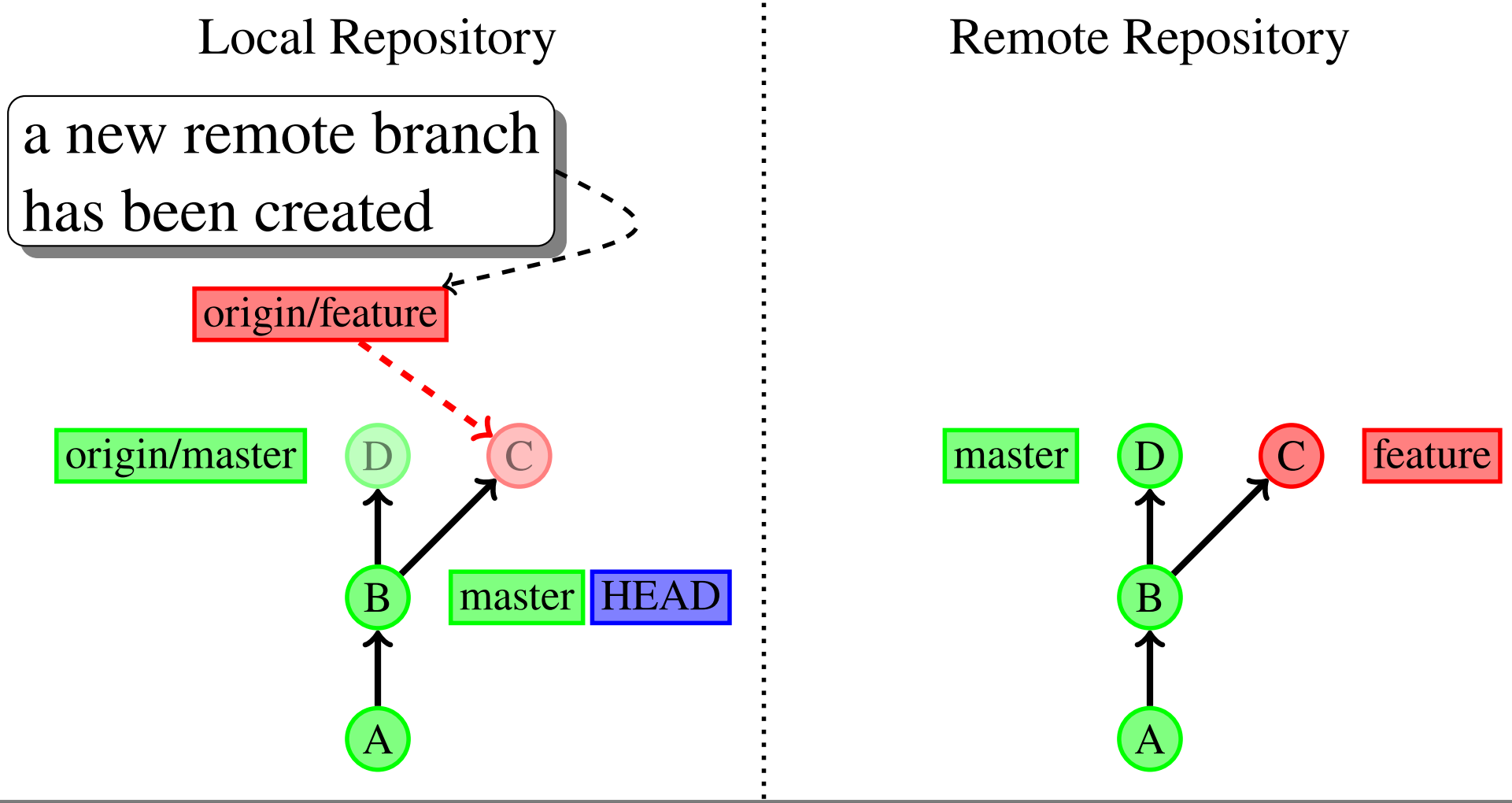


Remote Repository




```
git@test: git fetch origin
```

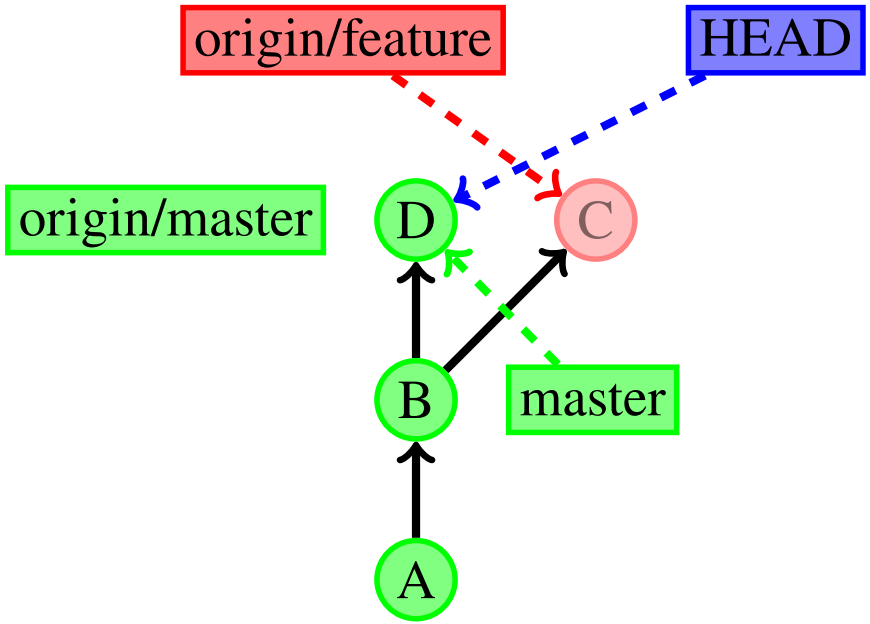
Working with branches - turning remote into local branches



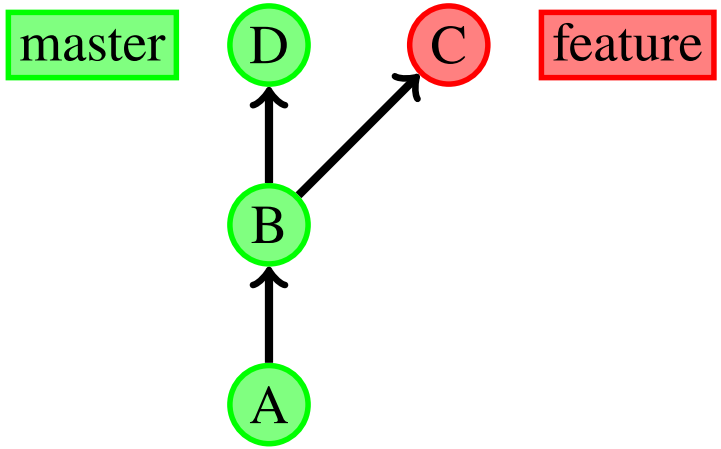

```
git@test: git fetch origin
```

Working with branches - turning remote into local branches

Local Repository

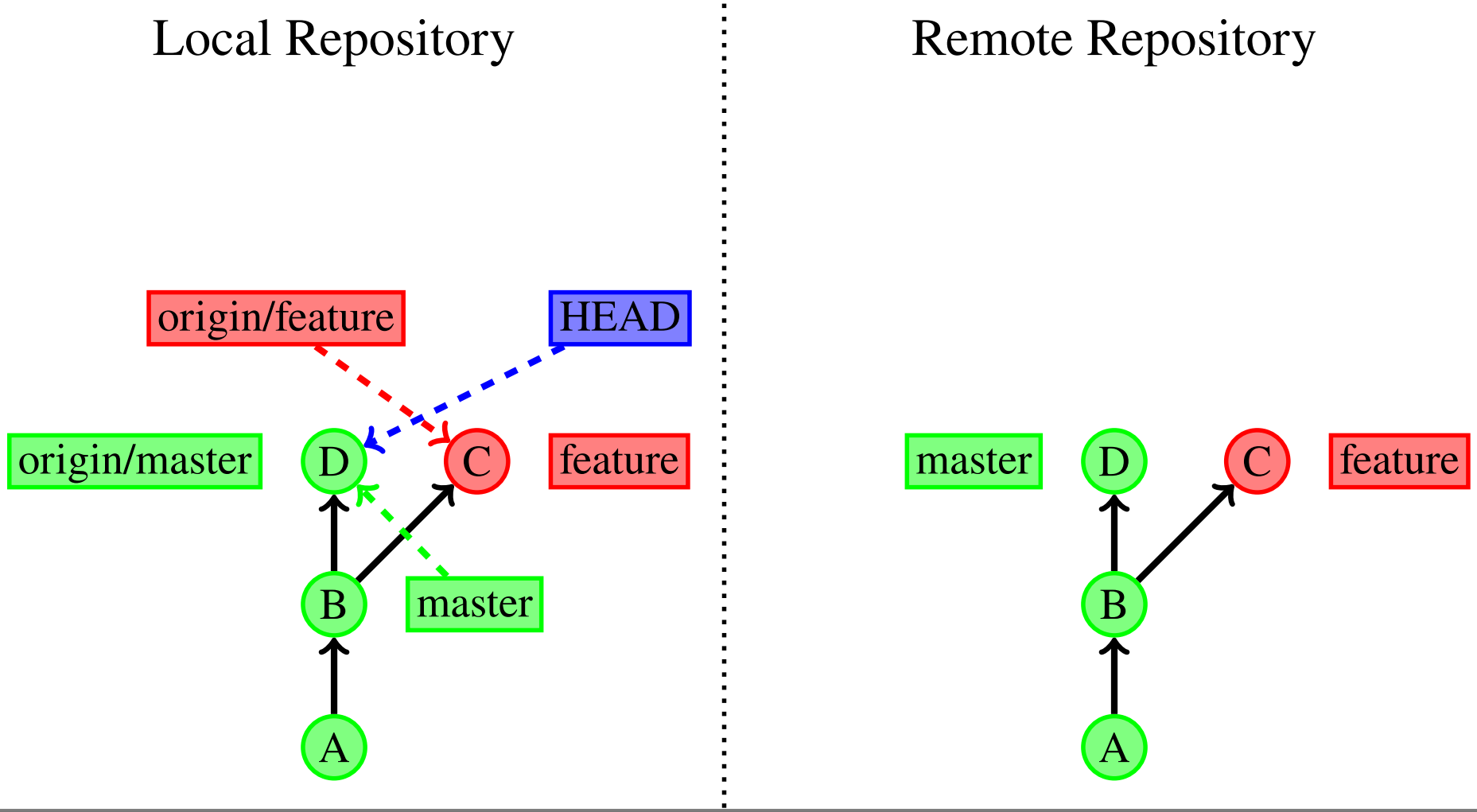


Remote Repository



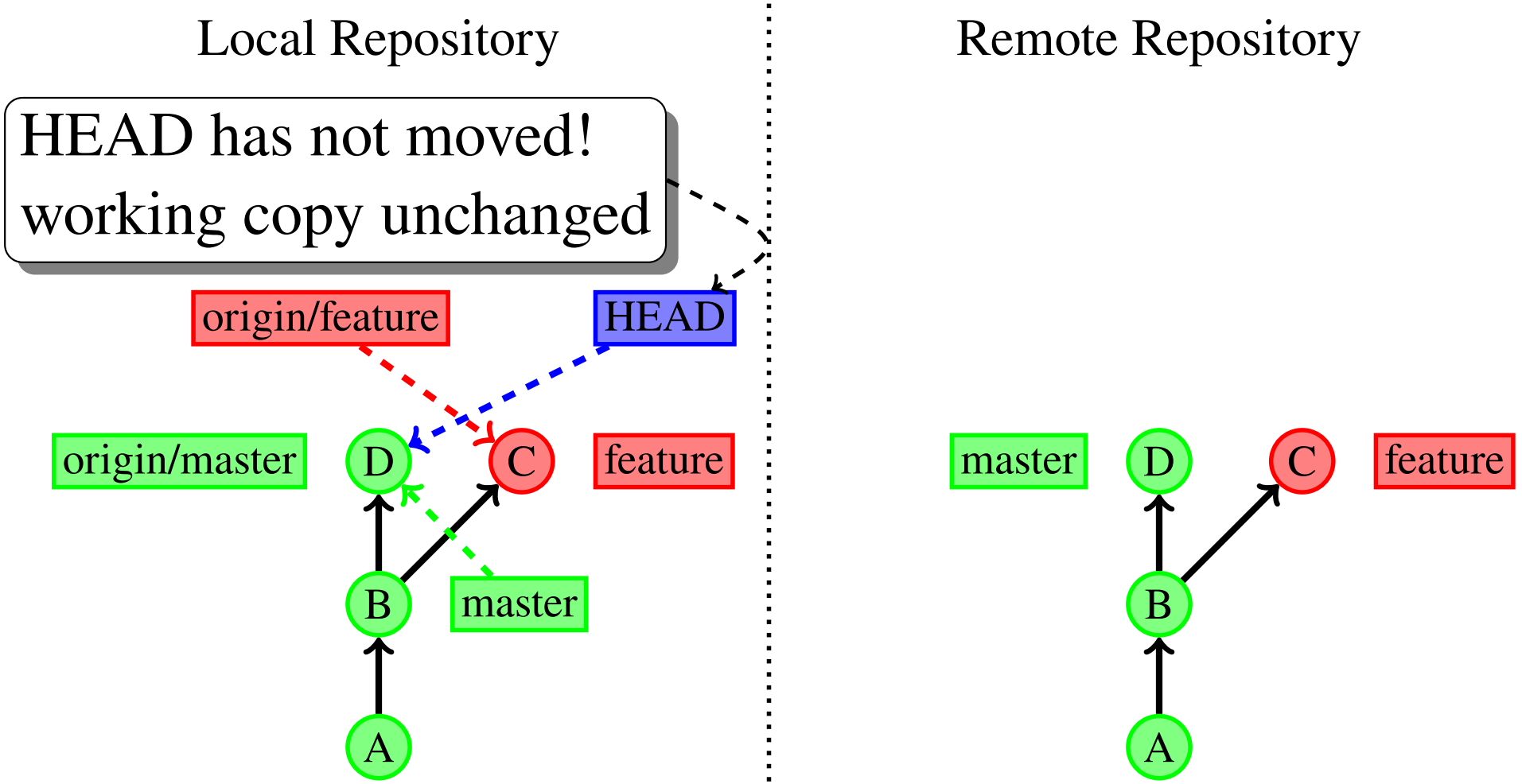

```
git@test: git fetch origin
git@test: git merge --ff-only origin/master
```

Working with branches - turning remote into local branches



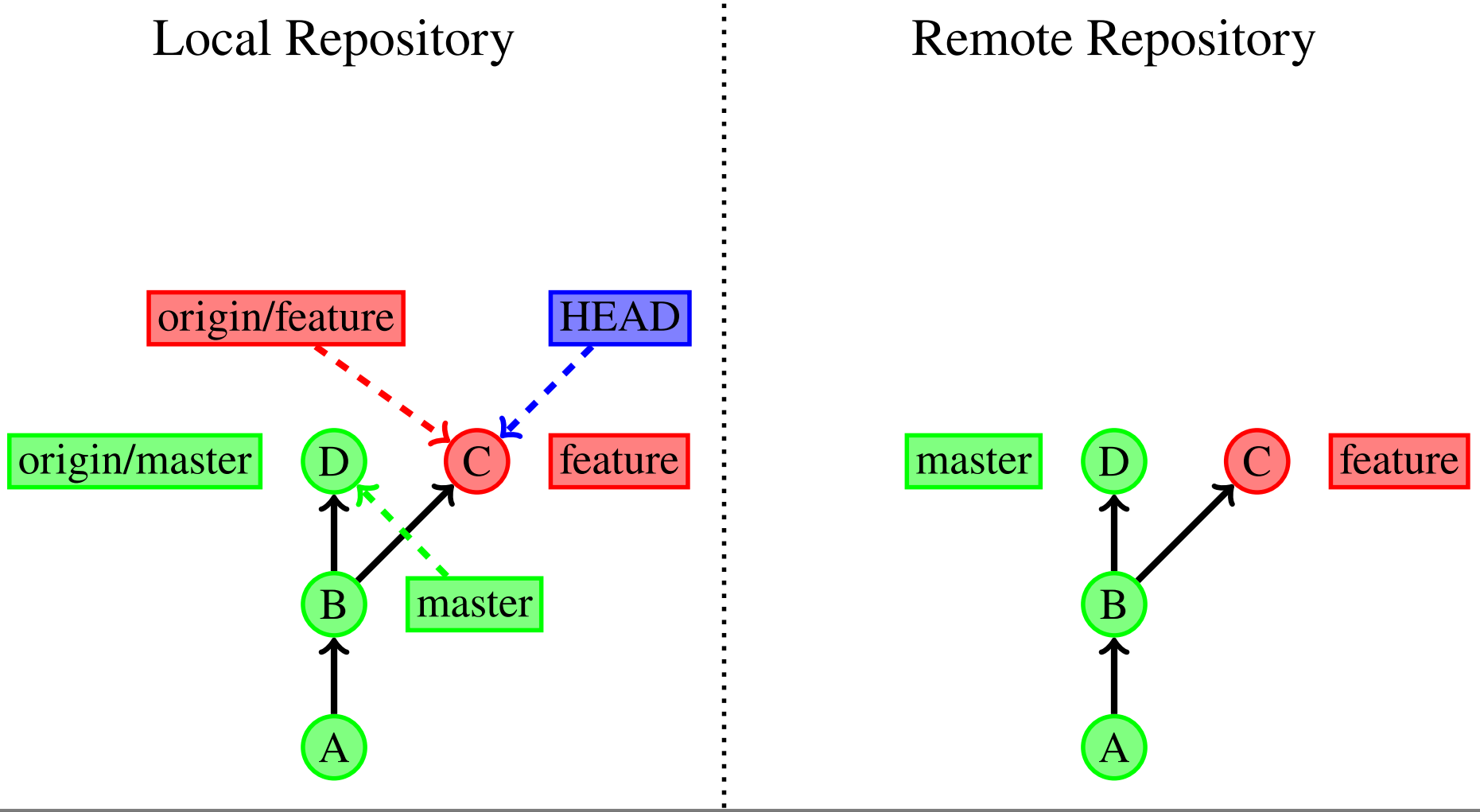

```
git@test: git fetch origin
git@test: git merge --ff-only origin/master
git@test: git branch feature origin/feature
```

Working with branches - turning remote into local branches




```
git@test: git fetch origin
git@test: git merge --ff-only origin/master
git@test: git branch feature origin/feature
```

Working with branches - turning remote into local branches



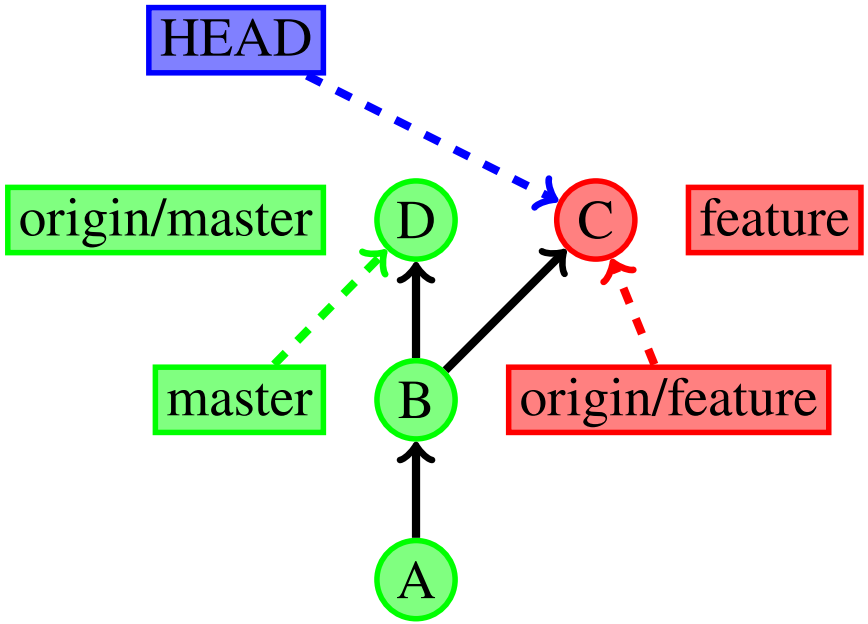

```
git@test: git fetch origin
git@test: git merge --ff-only origin/master
git@test: git branch feature origin/feature
git@test: git checkout feature
```

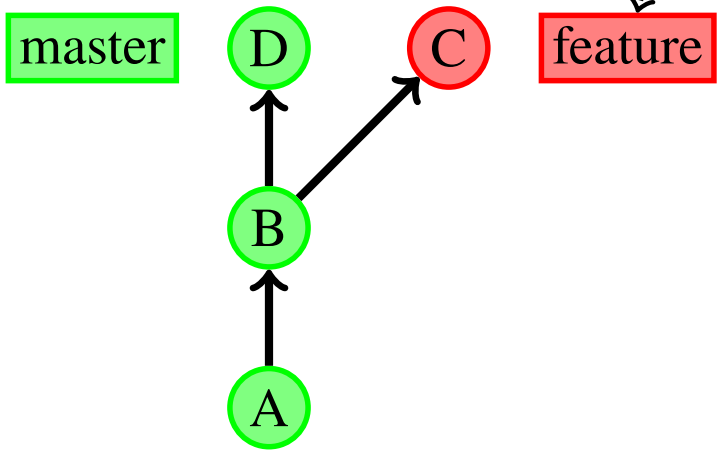
Working with branches I - merging finished branches

Local Repository

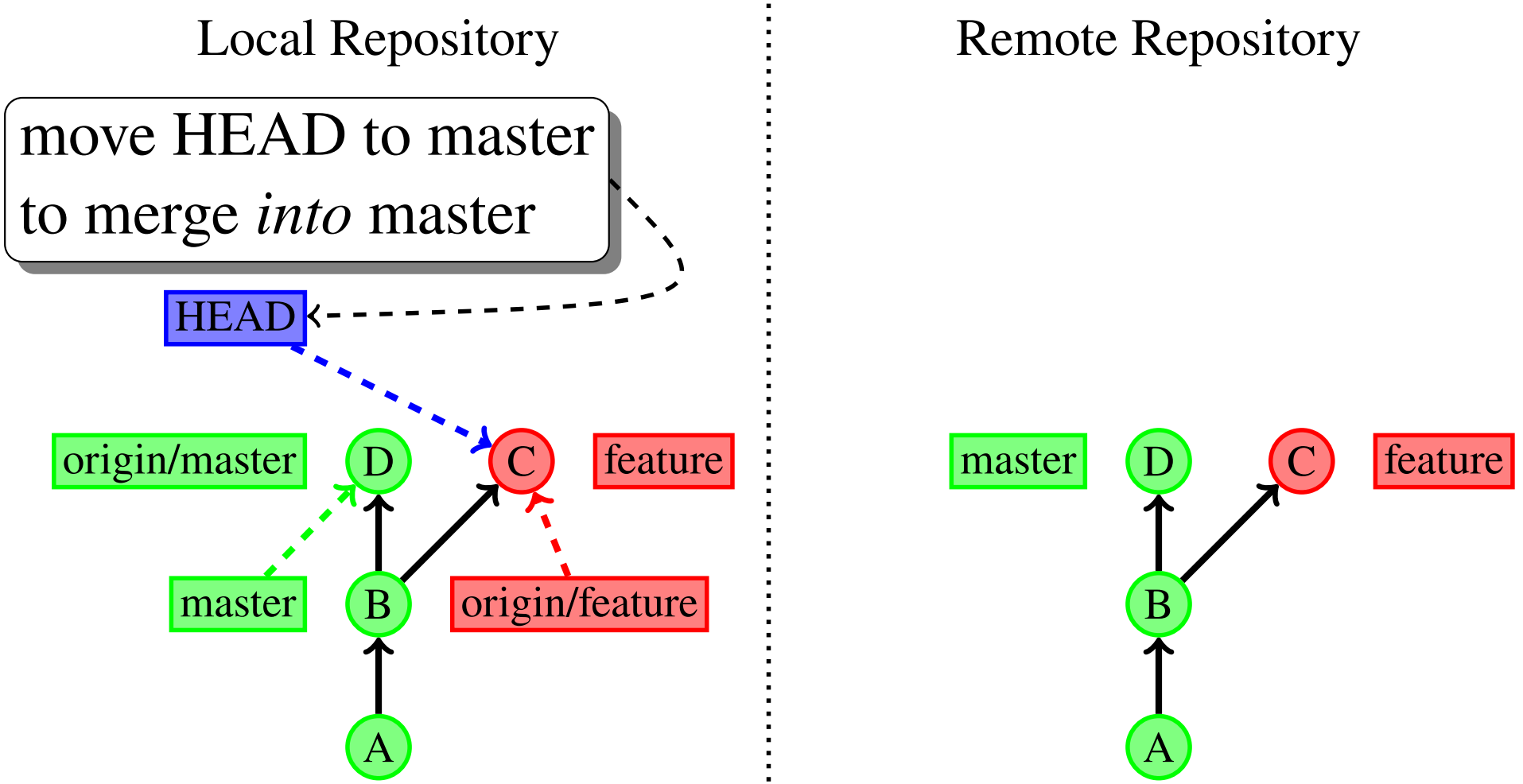
Remote Repository

Scenario I:
feature ready to merge

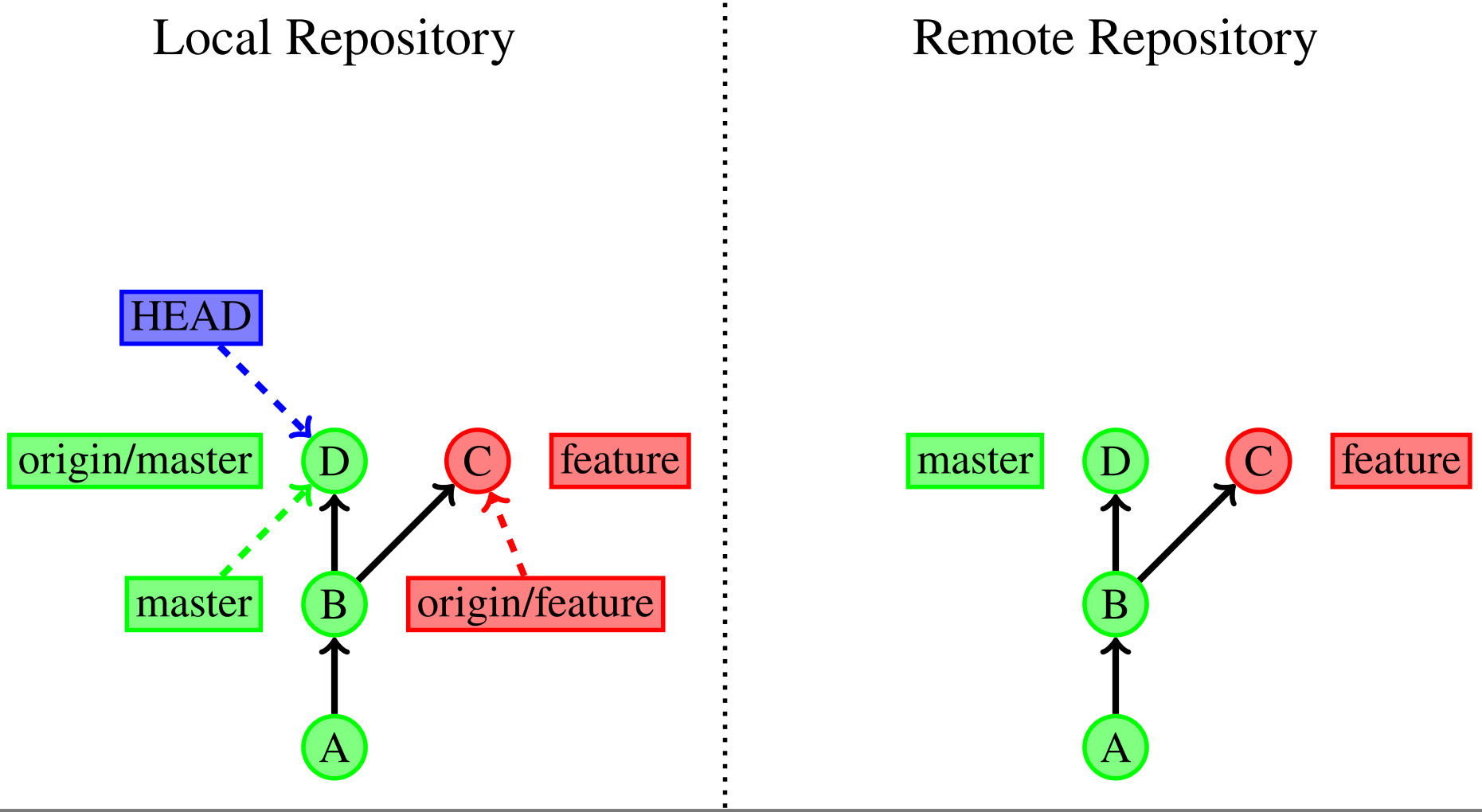




Working with branches I - merging finished branches



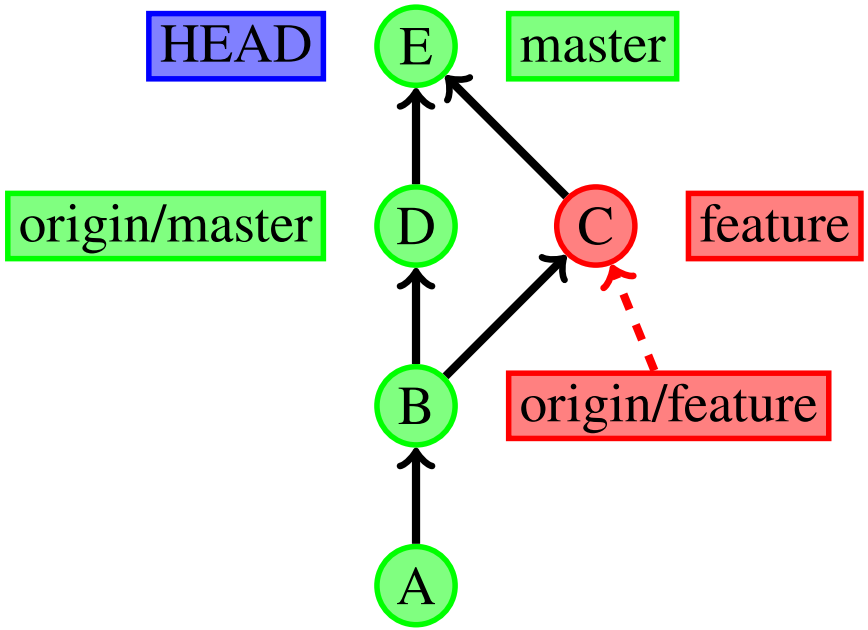
Working with branches I - merging finished branches



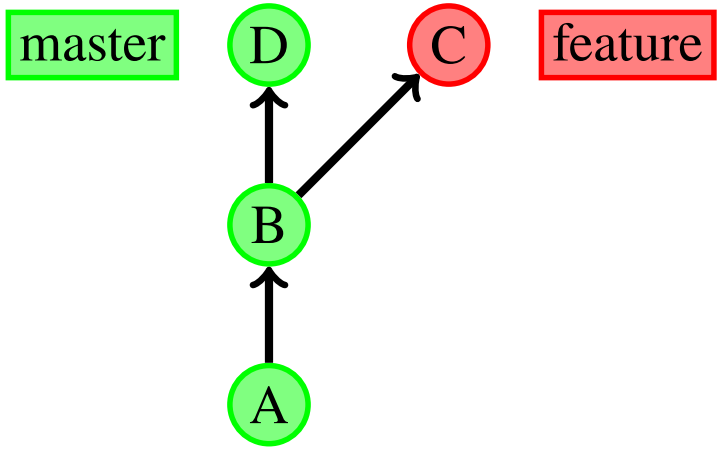

```
git@test: git checkout master
```

Working with branches I - merging finished branches

Local Repository



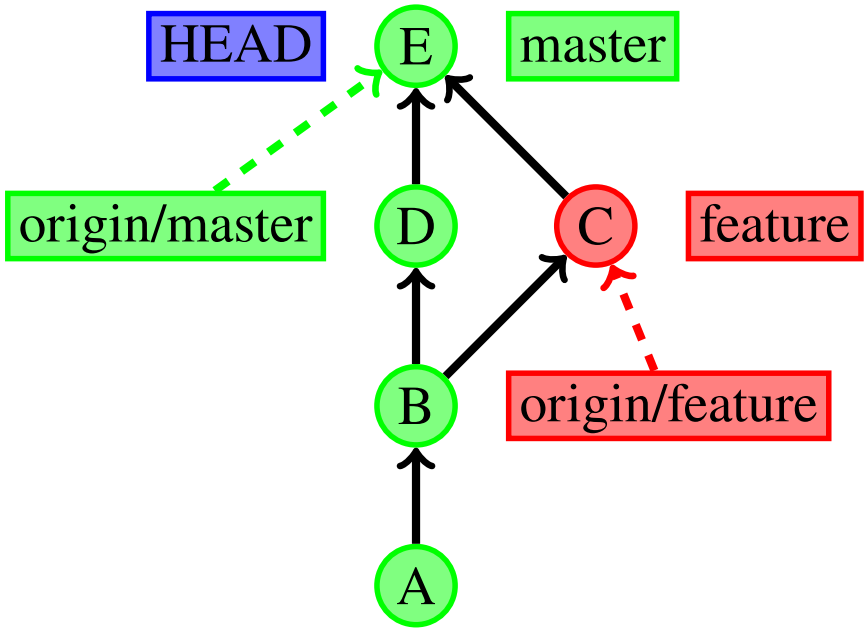
Remote Repository



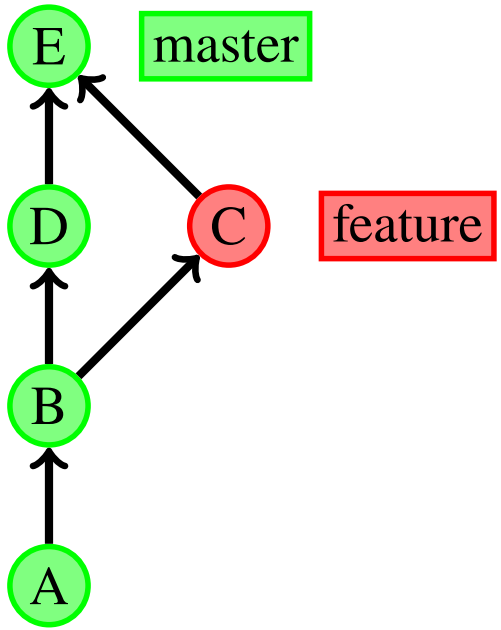

```
git@test: git checkout master
git@test: git merge feature
```

Working with branches I - merging finished branches

Local Repository



Remote Repository

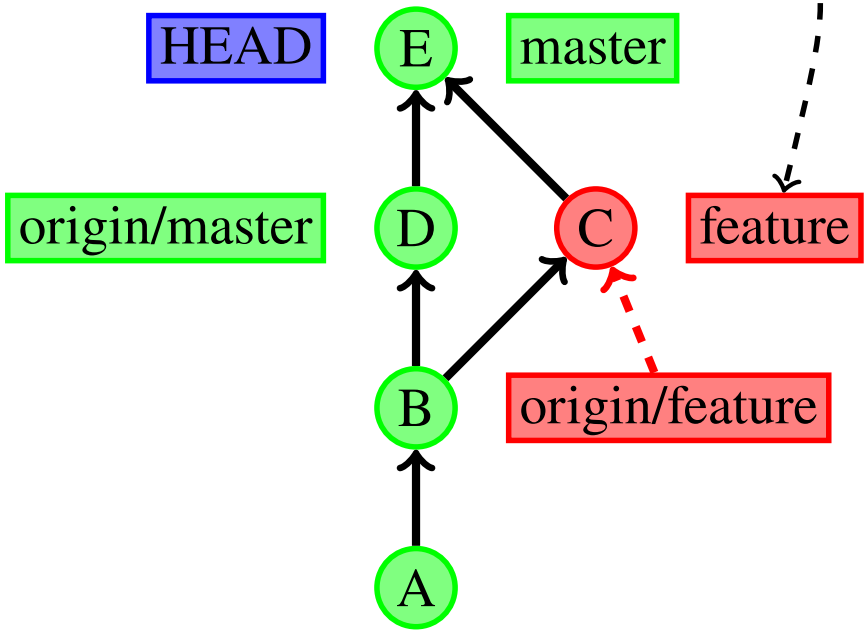



```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
```

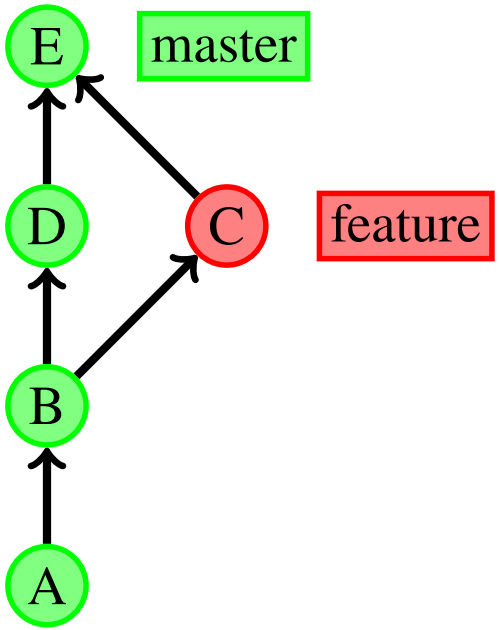
Working with branches I - merging finished branches

Local Repository

feature branch is no longer used, remove it



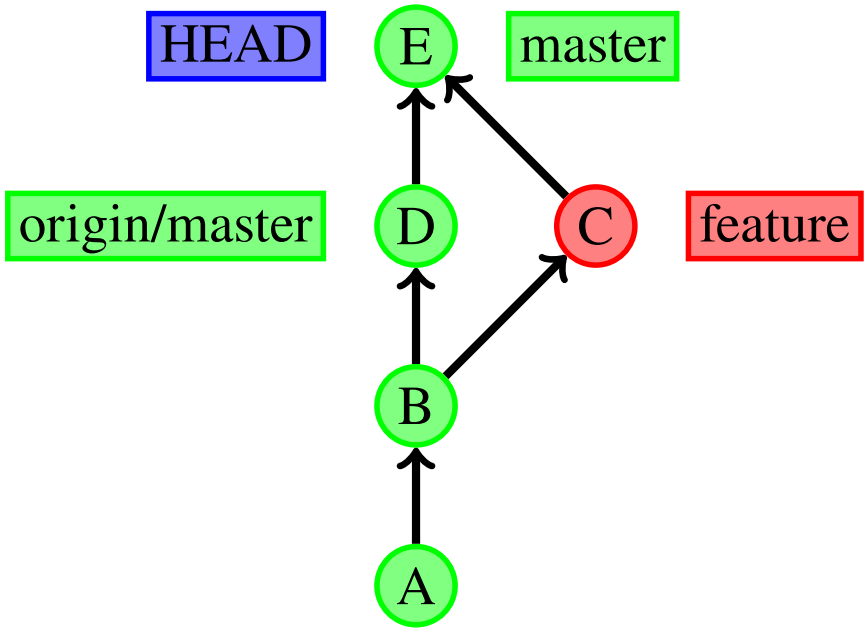
Remote Repository



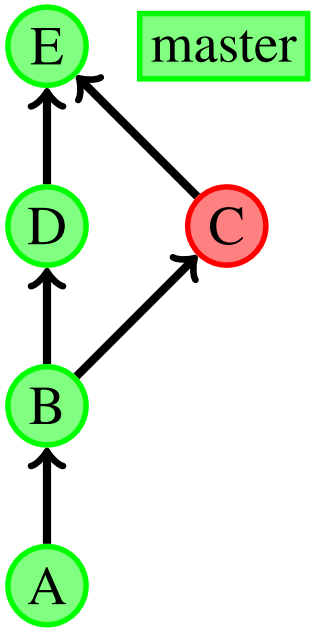

```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
```

Working with branches I - merging finished branches

Local Repository



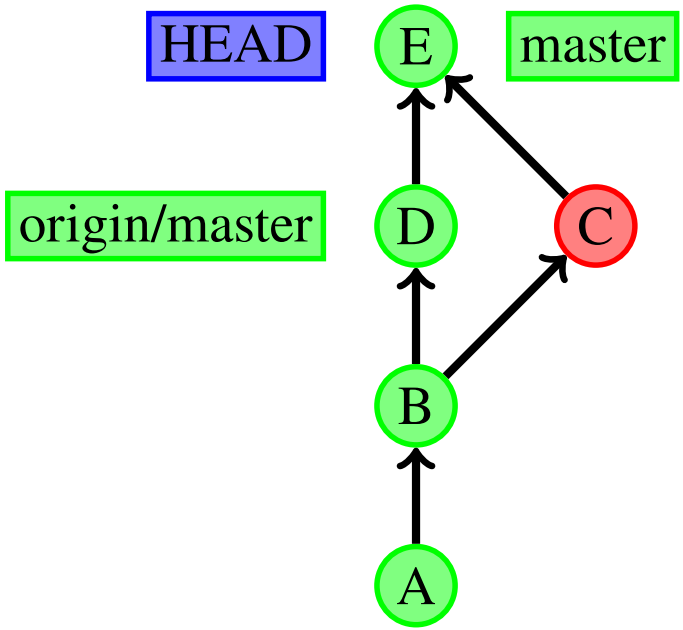
Remote Repository



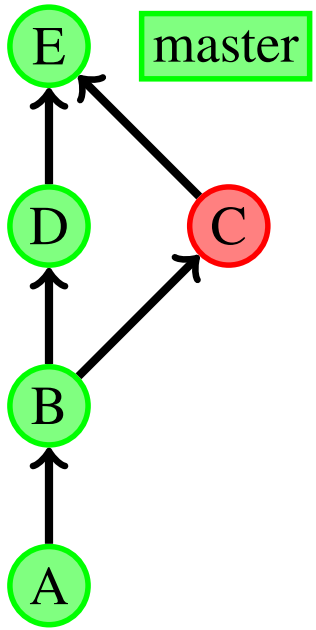

```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
git@test: git push --delete origin feature
```

Working with branches I - merging finished branches

Local Repository



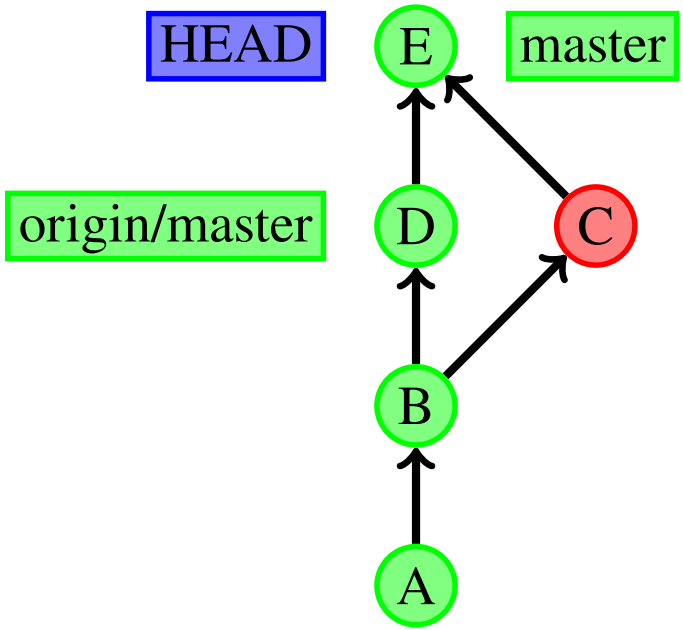
Remote Repository




```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
git@test: git push --delete origin feature
git@test: git branch -d feature
```

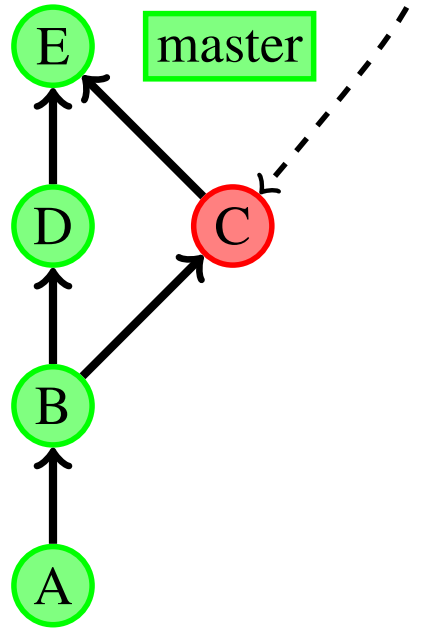
Working with branches I - merging finished branches

Local Repository



Remote Repository

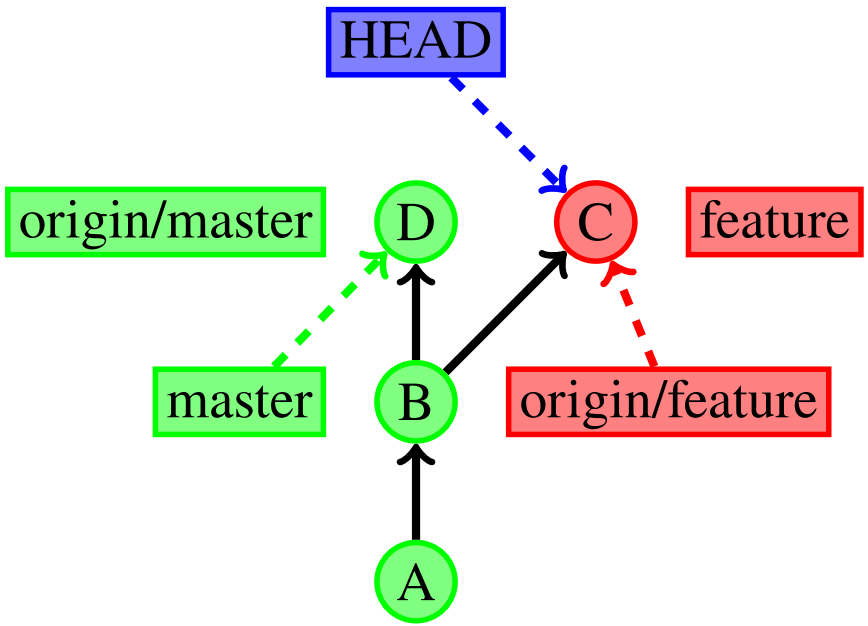
removing the branch causes no history loss




```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
git@test: git push --delete origin feature
git@test: git branch -d feature
```

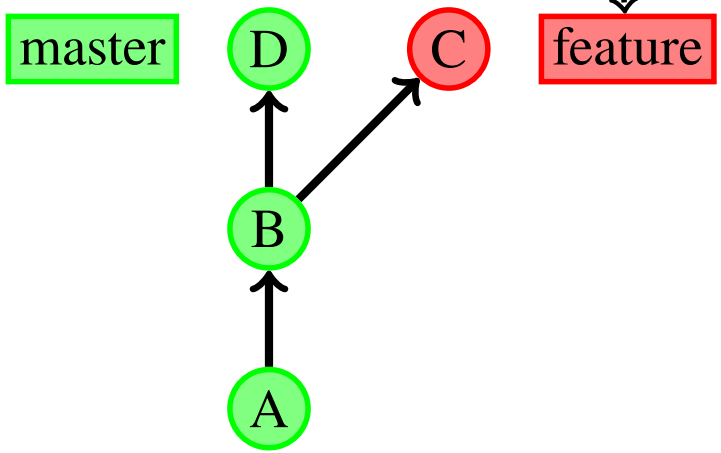
Working with branches II - updating feature branches

Local Repository



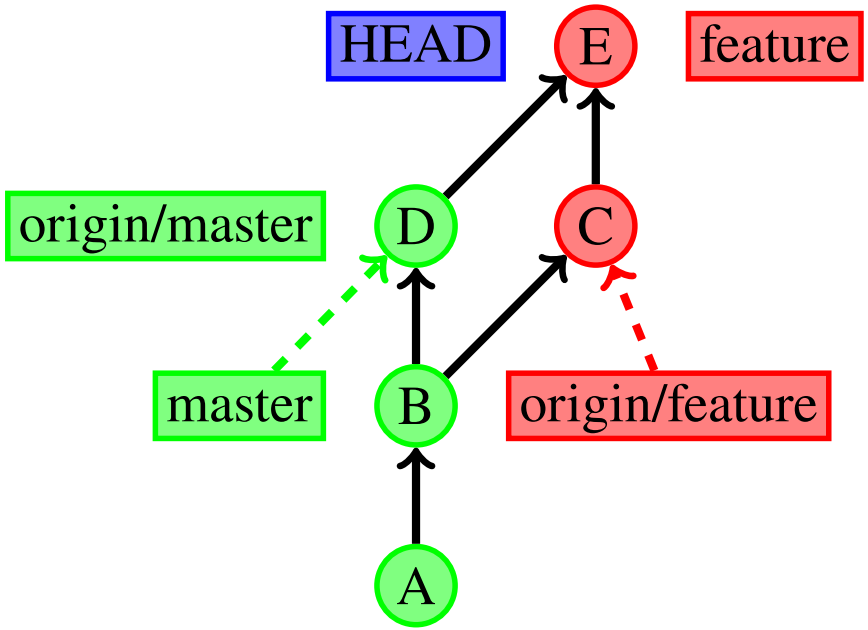
Remote Repository

Scenario II: feature needs new content from master branch

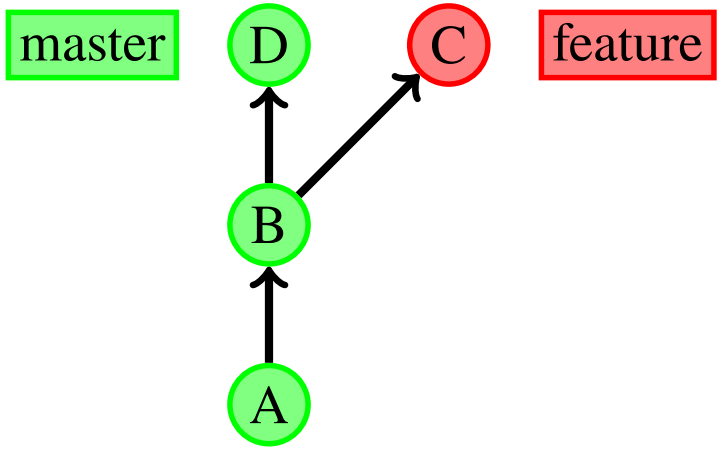


Working with branches II - updating feature branches

Local Repository



Remote Repository

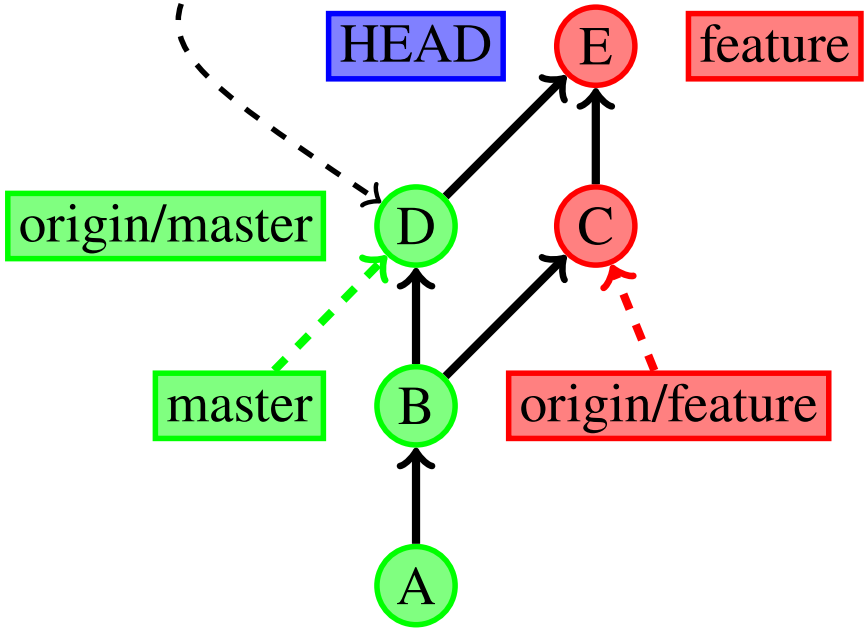



```
git@test: git merge master
```

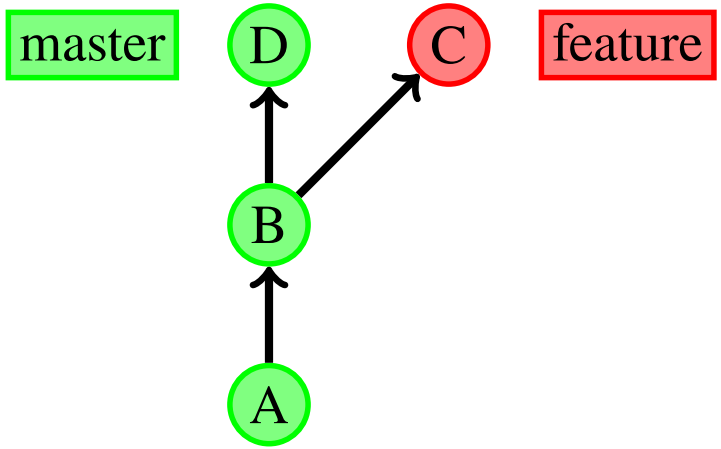
Working with branches II - updating feature branches

Local Repository

master is unchanged



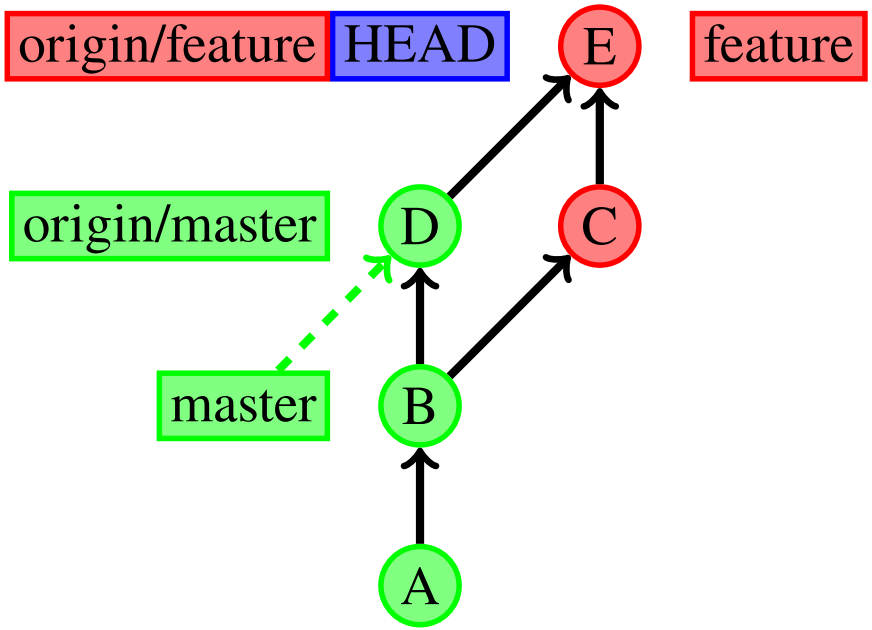
Remote Repository



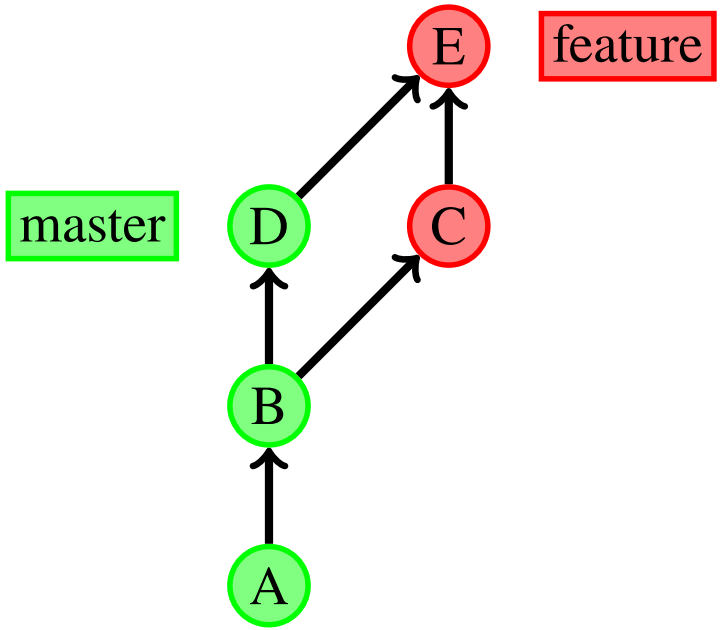

```
git@test: git merge master
```

Working with branches II - updating feature branches

Local Repository



Remote Repository




```
git@test: git merge master
git@test: git push origin feature
```

disaster recovery
how to clean the mess if you screwed up

As long as you did not commit…

`git checkout <revision> <path>`

set the content of the specified file(s) in the working copy to the given revision

`revision` the revision you want to restore, usually HEAD

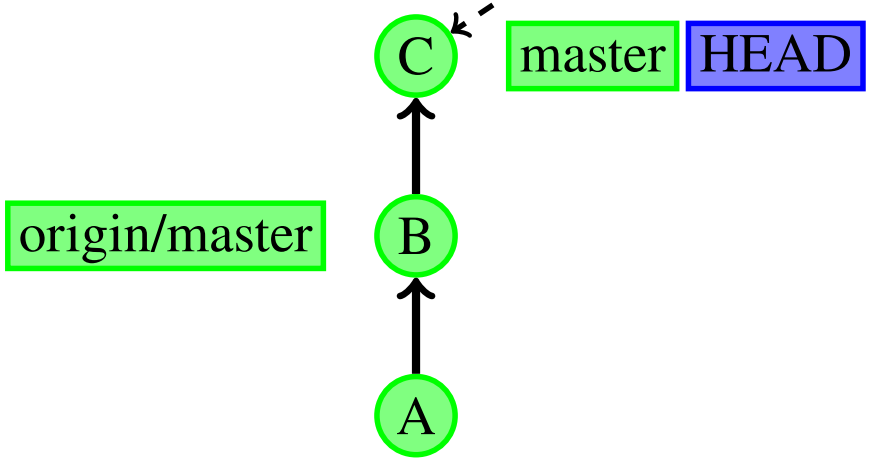
`path` shell pattern for the files to revert

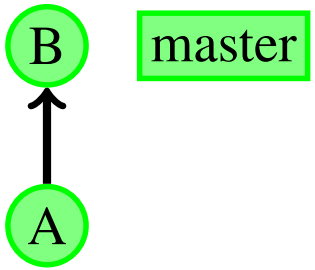
Undoing commits - deleting a local only commit

Local Repository

Remote Repository

Scenario I: this local commit is faulty





Undoing commits - deleting a local only commit

Local Repository | Remote Repository

`git reset [--hard] <revision>`

resets the branch tip to the specified revision
(while leaving the working directory unchanged)

`revision` the revision the tip should be set to, e.g. HEAD~1 to revert to the previous commit

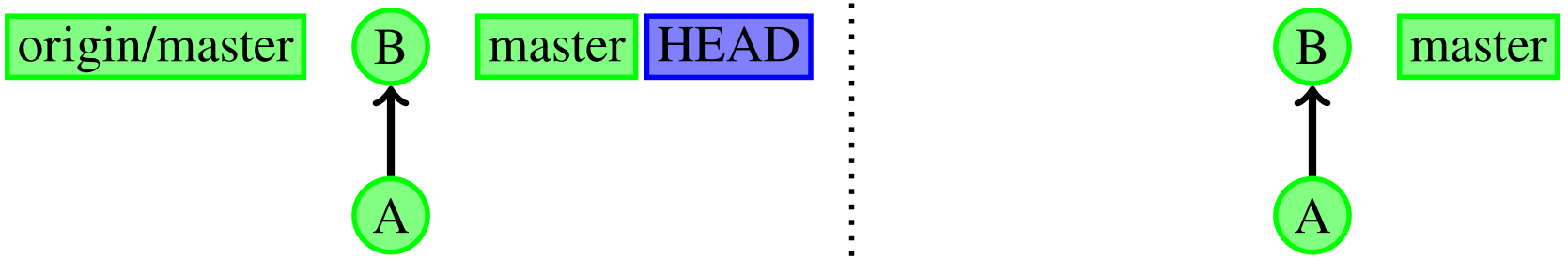
`--hard` purge the working directory and reset all files to their state in the target revision


```
git@test: git reset --hard HEAD~1
```

Undoing commits - deleting a local only commit

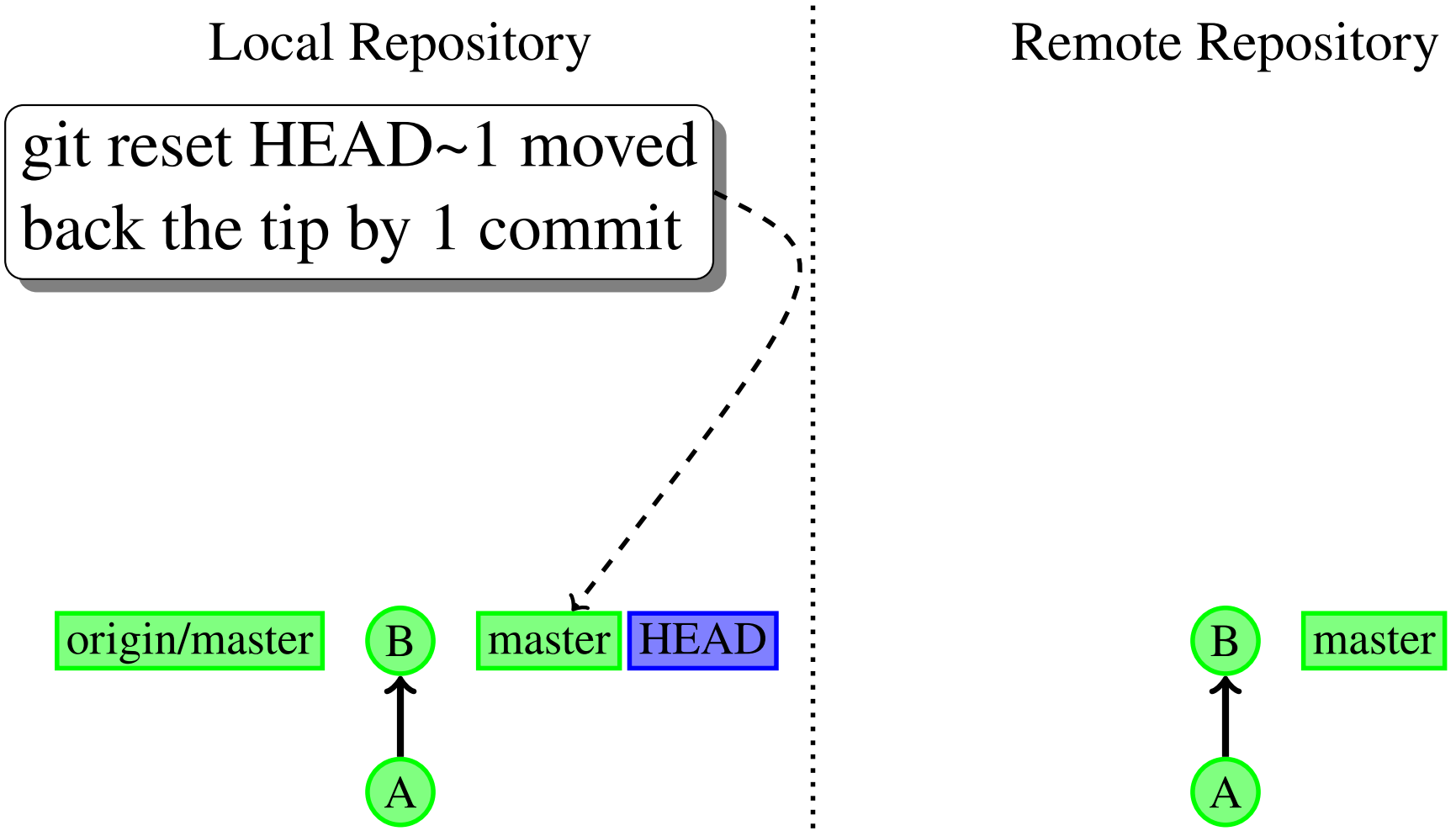
Local Repository

Remote Repository




```
git@test: git reset --hard HEAD~1
```

Undoing commits - deleting a local only commit



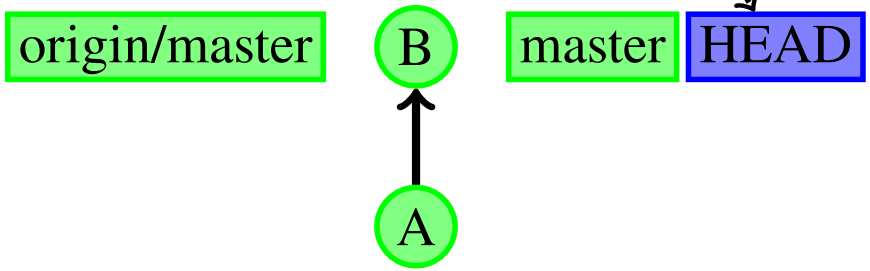

```
git@test: git reset --hard HEAD~1
```

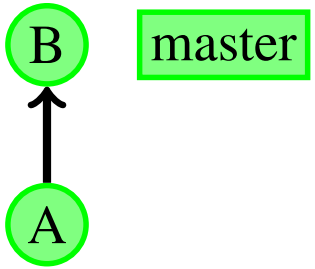
Undoing commits - deleting a local only commit

Local Repository

Remote Repository

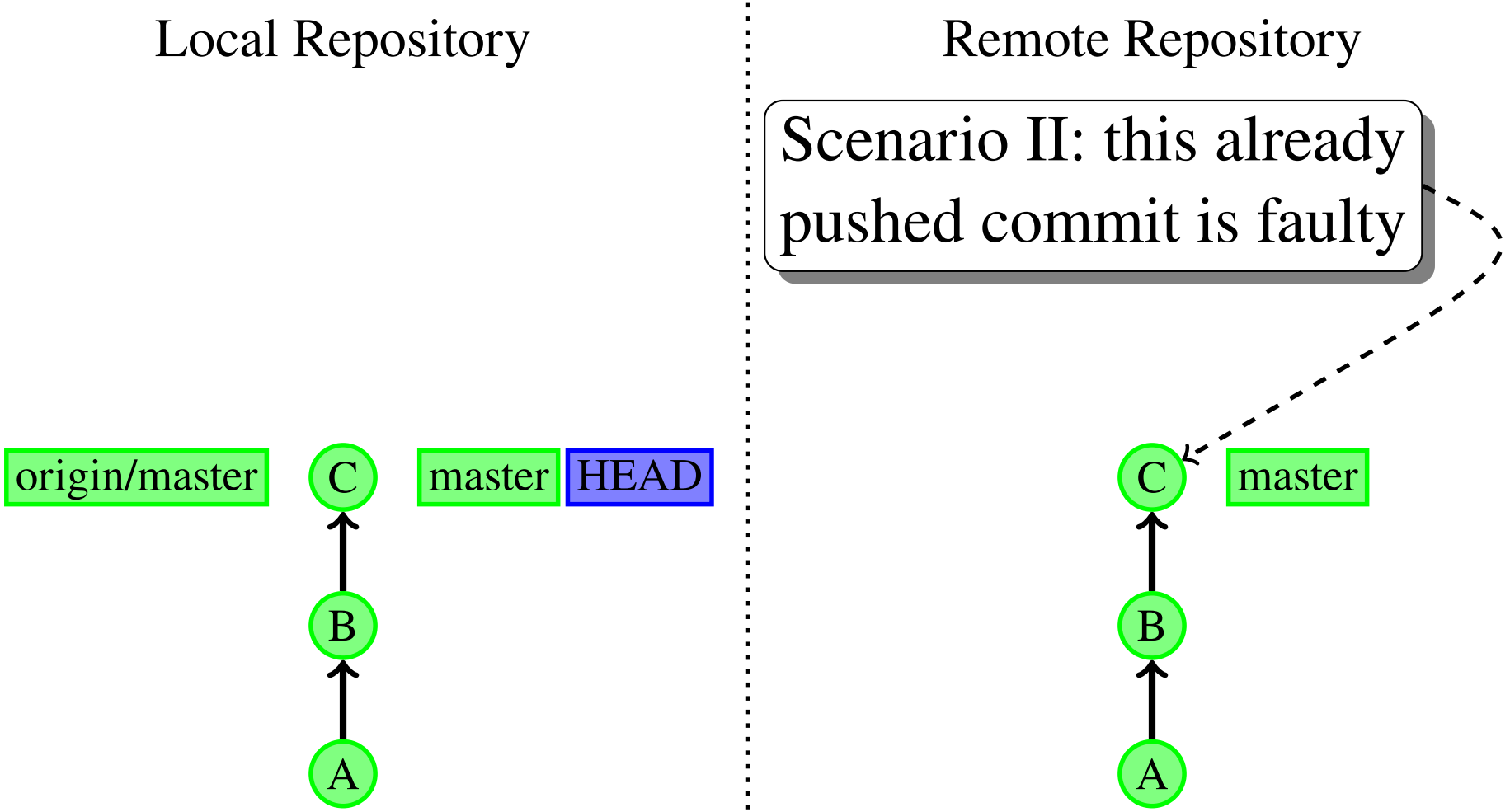
`--hard` resets all files
to their state in this commit



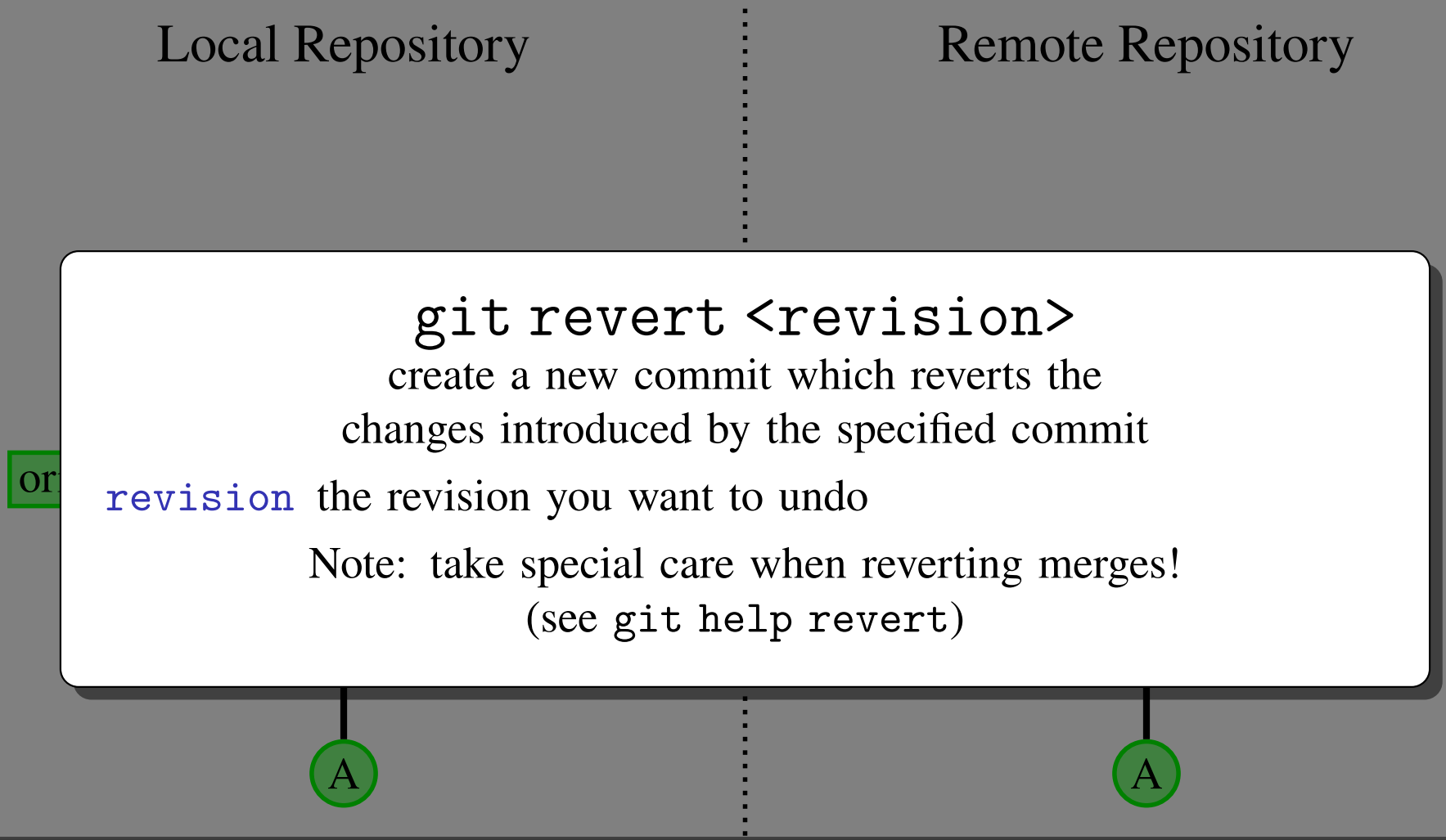



```
git@test: git reset --hard HEAD~1
```

Undoing commits - reverting an already pushed commit

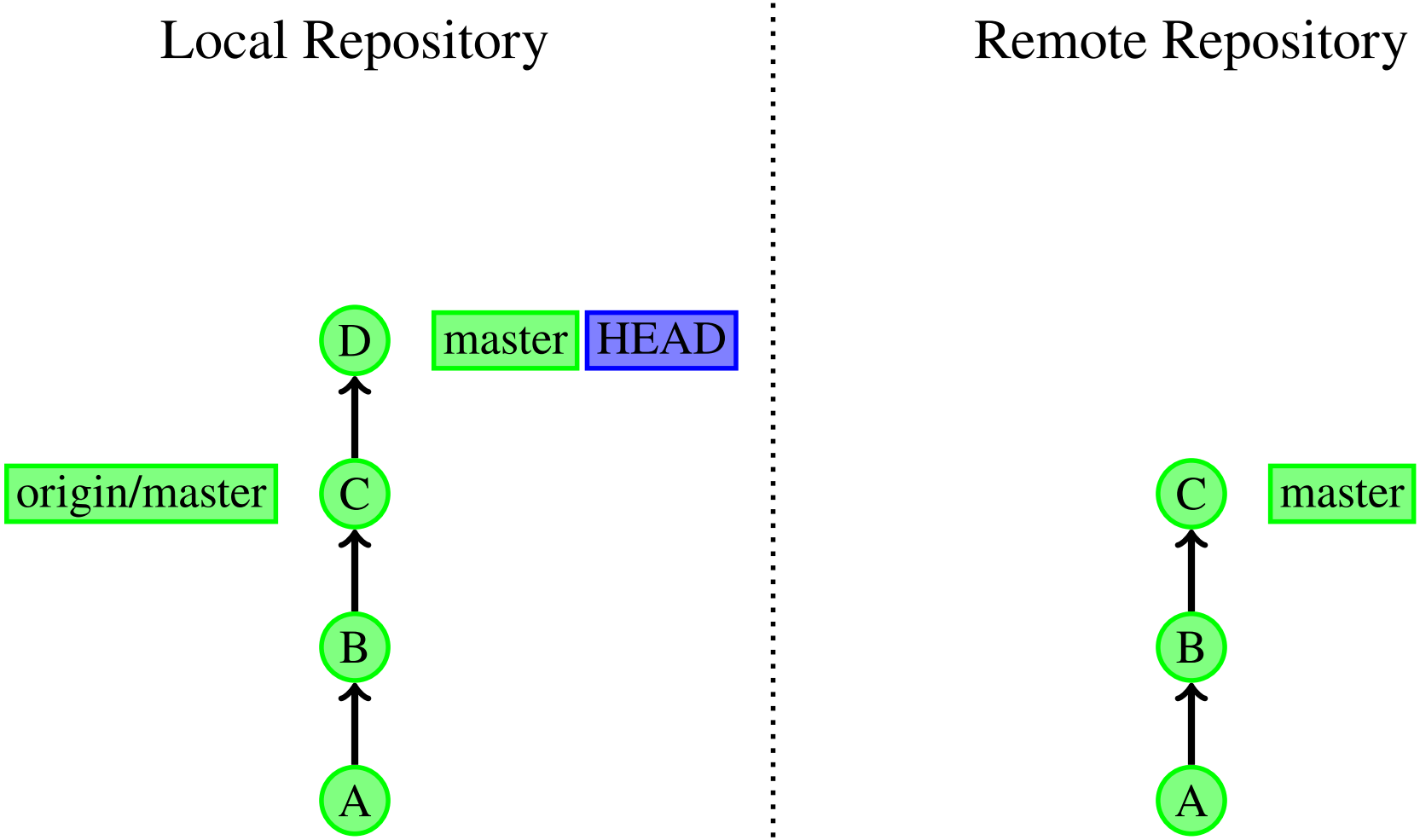


Undoing commits - reverting an already pushed commit



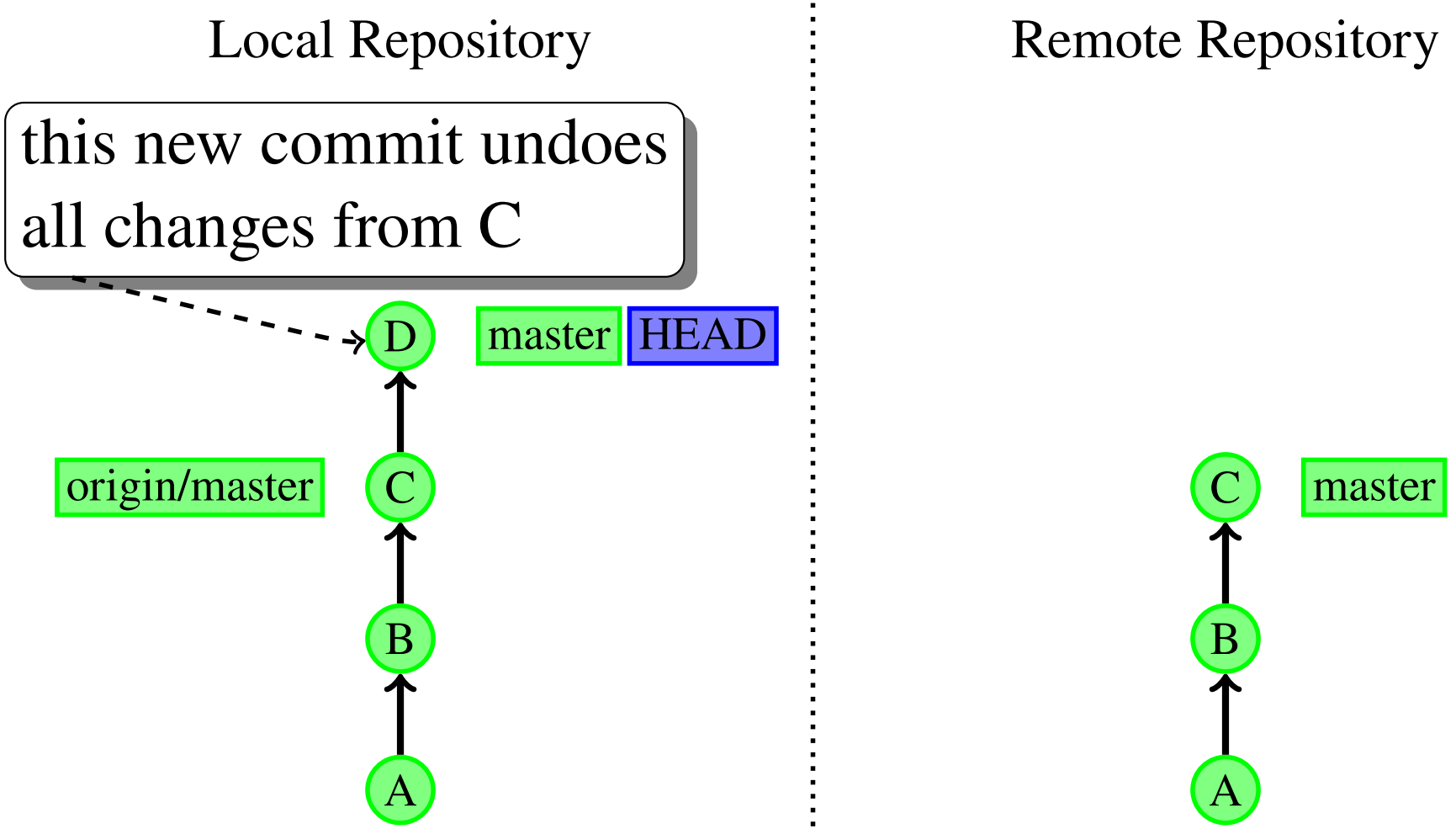

```
git@test: git revert HEAD
```

Undoing commits - reverting an already pushed commit




```
git@test: git revert HEAD
```

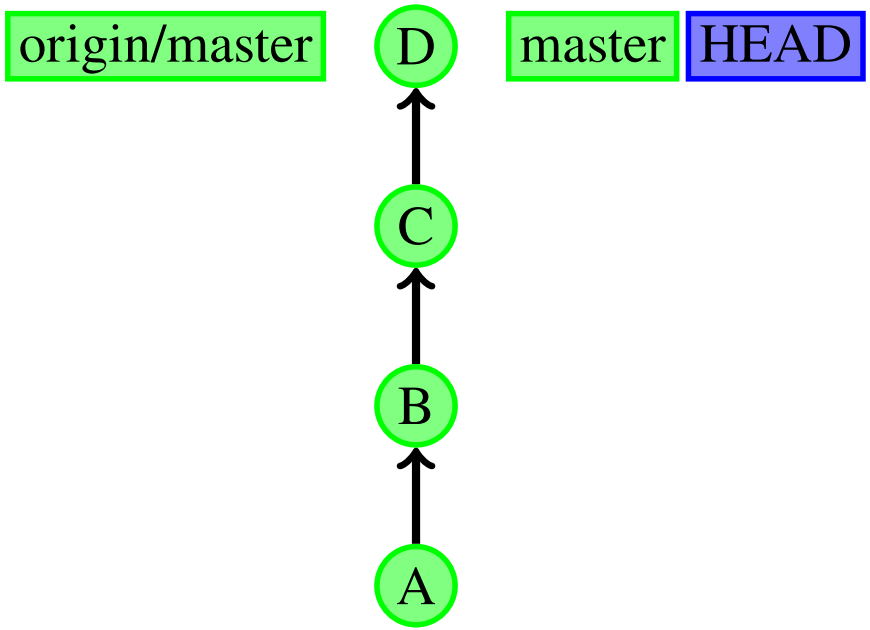
Undoing commits - reverting an already pushed commit



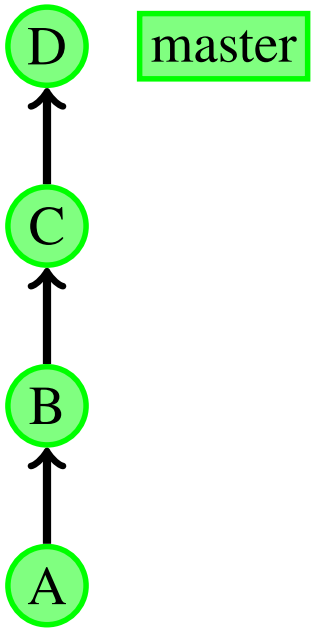

```
git@test: git revert HEAD
```

Undoing commits - reverting an already pushed commit

Local Repository



Remote Repository




```
git@test: git revert HEAD
git@test: git push origin master
```

recovering from disaster recovery
how to survive screwing up disaster recovery…

`git reflog [<branch>]`

show the revision log for the specified branch, i.e. all movements of the branch tip

HEAD's reflog also records branch switches

`branch` the branch for which the reflog should be shown, defaults to HEAD

`git help [<command>]`

show the detailed manpage for a command

`command` the command of interest

GUI alternatives to the command line

I don’t like the terminal, can I use a GUI?

GUI alternatives to the command line

I don’t like the terminal, can I use a GUI?

- Linux: gitg, gitk, qgit

GUI alternatives to the command line

I don’t like the terminal, can I use a GUI?

- Linux: gitg, gitk, qgit
- Mac and Windows: Atlassian Sourcetree

```
git commit -m "This is the end of the presentation. Thank you for your attention!"
```