



# git

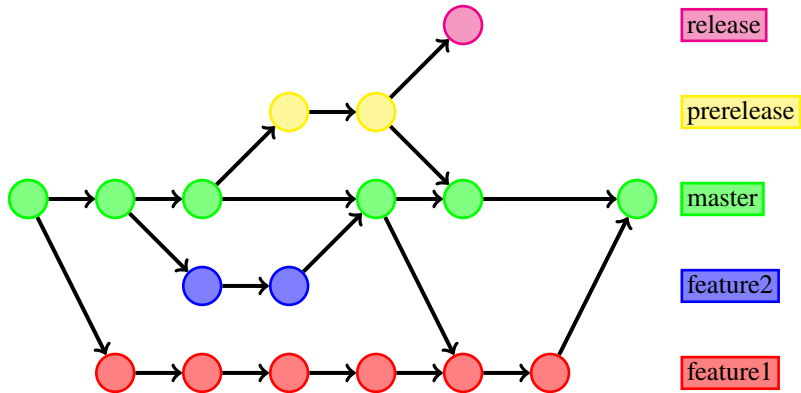
an introduction to the power of "the stupid content tracker"

Arvid Conrad Ihrig

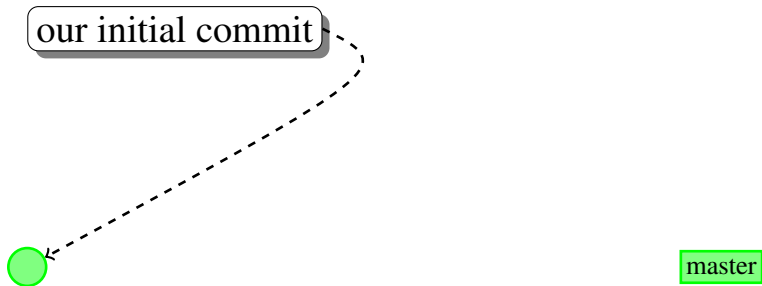
Fritz-Haber-Institut der Max-Planck-Gesellschaft

August 19th 2014, Berlin

# The Concept of Distributed Version Control

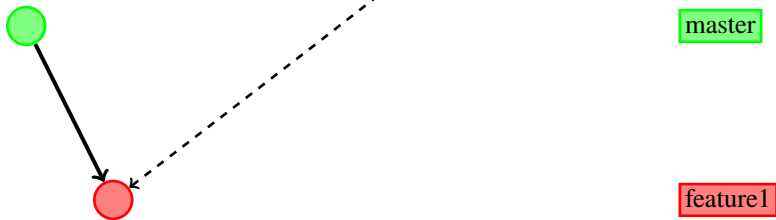


# The Concept of Distributed Version Control

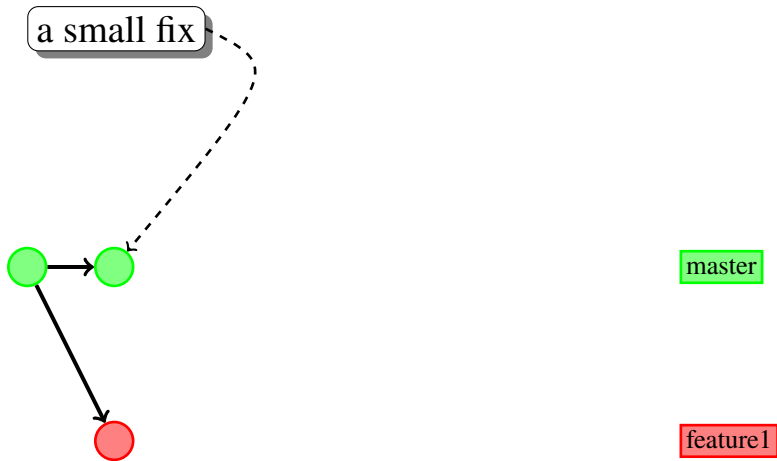


# The Concept of Distributed Version Control

branching of a complex feature

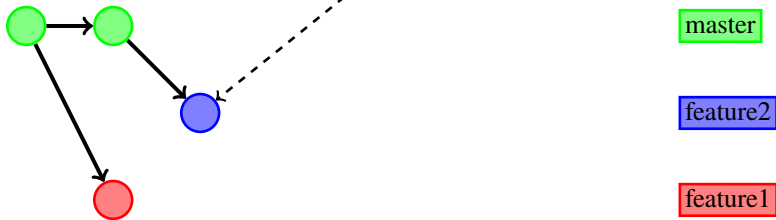


# The Concept of Distributed Version Control



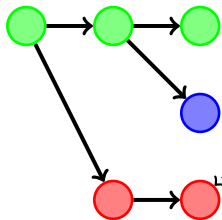
# The Concept of Distributed Version Control

branching of another feature



# The Concept of Distributed Version Control

normal development commits



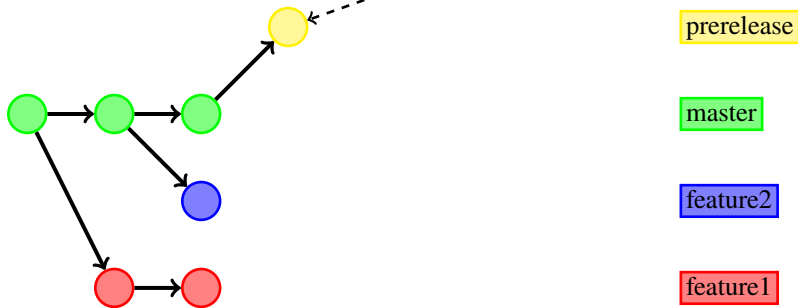
master

feature2

feature1

# The Concept of Distributed Version Control

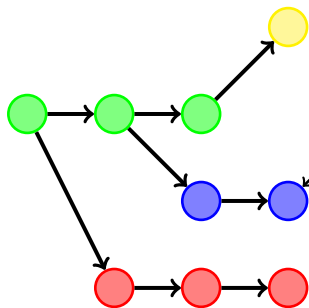
feature freeze before the release





# The Concept of Distributed Version Control

normal development commits



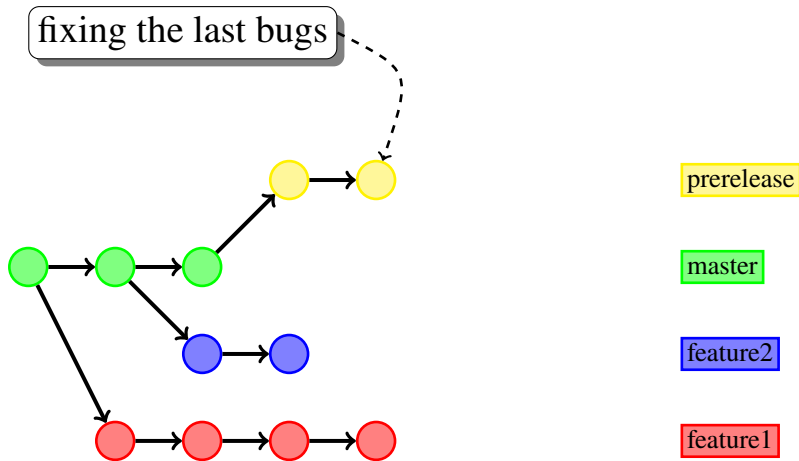
prerelease

master

feature2

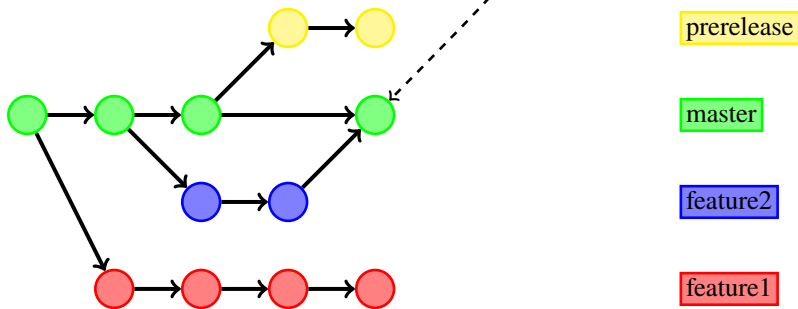
feature1

# The Concept of Distributed Version Control



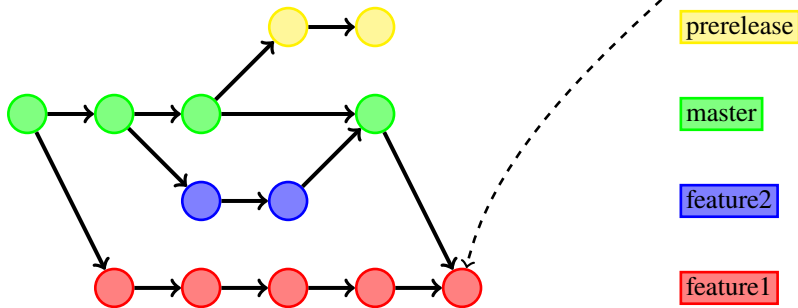
# The Concept of Distributed Version Control

merge back a finished feature

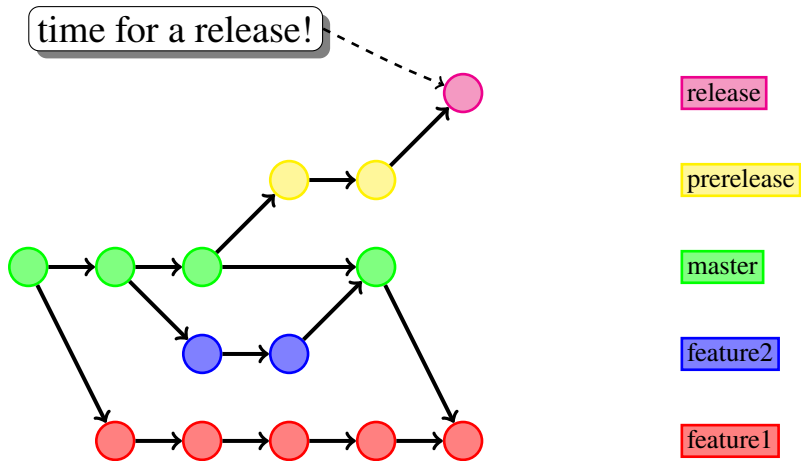


# The Concept of Distributed Version Control

merge from master to keep feature up-to-date

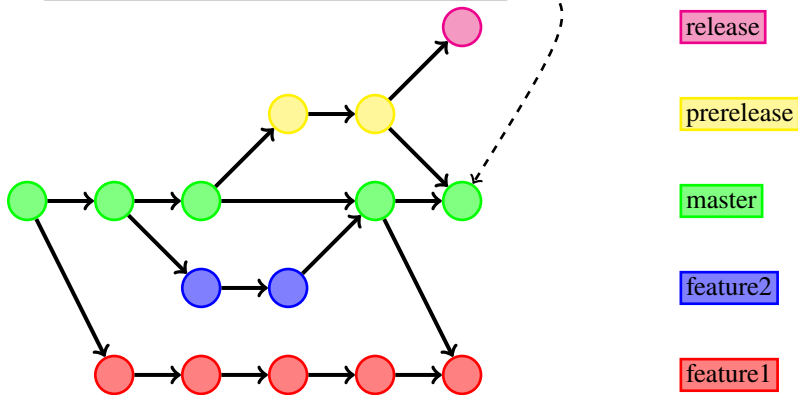


# The Concept of Distributed Version Control



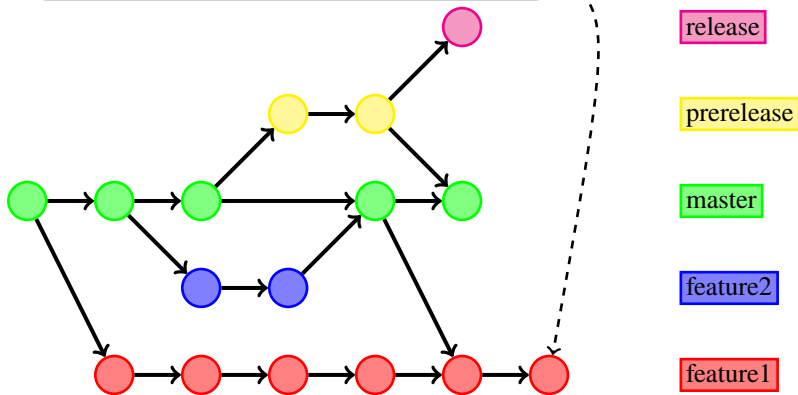
# The Concept of Distributed Version Control

merge back the small bugfixes



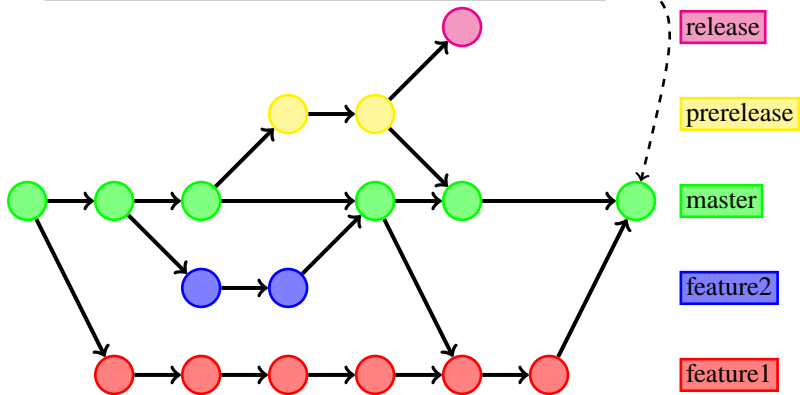
# The Concept of Distributed Version Control

final changes to complex feature



# The Concept of Distributed Version Control

merge complex feature into mainline

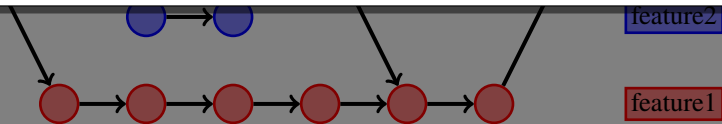




# The Concept of Distributed Version Control

## 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

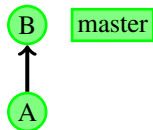
## Basic git usage

recording changes, inspecting commits and remote  
repositories

# Cloning an existing git repository

Local Repository

Remote Repository



# Cloning an existing git repository

Local Repository

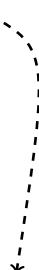
Remote Repository

server branch tip

B

master

A



# 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

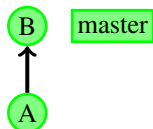
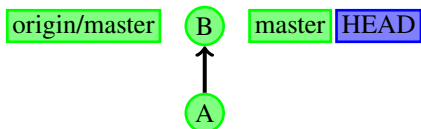


```
git@test: git clone user@server:path/to/repo
```

# Cloning an existing git repository

Local Repository

Remote Repository



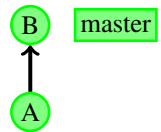
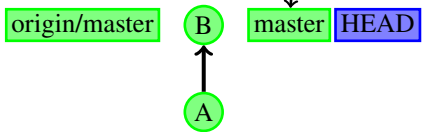
```
git@test: git clone user@server:path/to/repo
```

# Cloning an existing git repository

Local Repository

Remote Repository

local branch tip



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

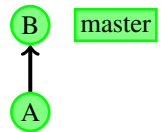
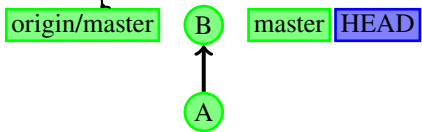


# Cloning an existing git repository

Local Repository

Remote Repository

remote branch tip



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

# Cloning an existing git repository

Local Repository

Remote Repository

state of your  
working directory

origin/master

B



master

HEAD

B

master



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

# Cloning an existing git repository

Local Repository

Remote Repository

```
git show <revision>
```

show the given commit and the contained changes

`revision` the commit you want to inspect

origin/master

B

master HEAD



B

master

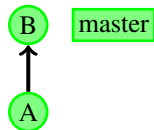
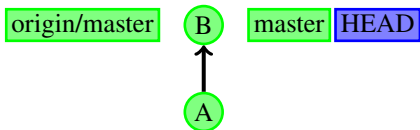


```
git@test: git clone user@server:path/to/repo
```

# Committing your changes locally

Local Repository

Remote Repository



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

# Committing your changes locally

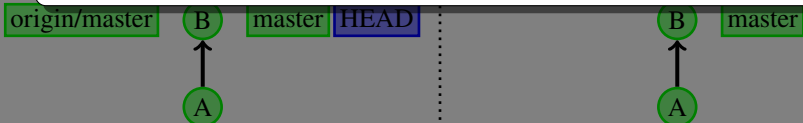
Local Repository

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

Local Repository

Remote Repository

```
git add [-p] <path>
```

mark the changes in the specified file(s) as part of the next commit

**path** path specification for the file(s) to include

**-p** start an interactive session, where you can decide for each change in a file individually if it should be committed

or

A

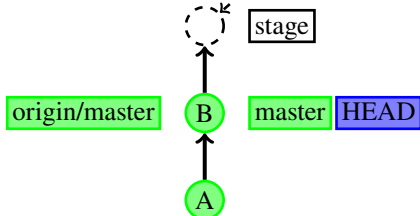
A

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

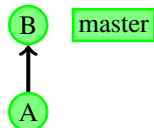
# Committing your changes locally

Local Repository

the changed file  
is now *staged* for  
the next commit



Remote Repository



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

# Committing your changes locally

Local Repository

Remote Repository

```
git commit [--amend]
```

save the previously staged changes as a new commit after entering a commit message

`--amend` instead of making a new commit, correct the current HEAD, e.g. to fix typos or include forgotten files (use only for not pushed commits!)

or

A

A

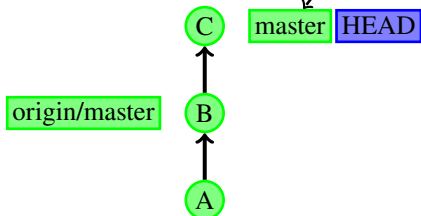
```
git@test: echo "This should be fun!" > newfile.txt
git@test: git status
git@test: git add newfile.txt
git@test: git commit
```



# Committing your changes locally

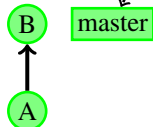
Local Repository

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

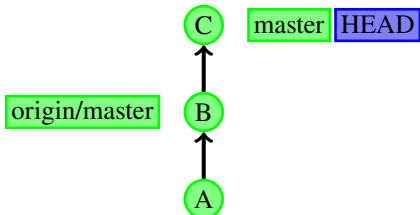
or `path` shell pattern for file(s) to diff

`--cached` use the staged content as `revision1`

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

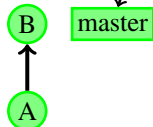
# Uploading your changes I - no conflicts

Local Repository



Remote Repository

scenario I: nothing new on the server



# 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

origin

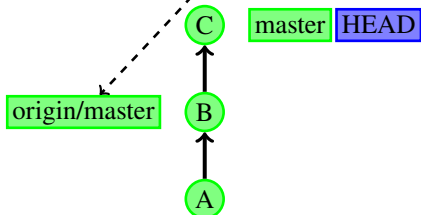


```
git@test: git fetch origin
```

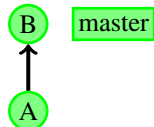
# Uploading your changes I - no conflicts

Local Repository

remote branch  
stays the same



Remote Repository

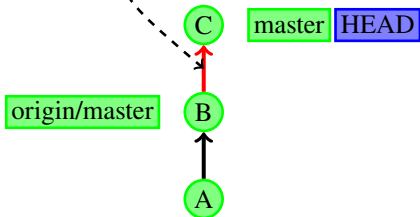


```
git@test: git fetch origin
```

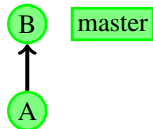
# Uploading your changes I - no conflicts

Local Repository

linear history connection  
"fast forward" possible



Remote Repository



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

# Uploading your changes I - no conflicts

Local Repository

Remote Repository

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

push the specified branch to the given remote,  
i.e. upload all new commits in this branch

`remote` the remote repository to push to

`branch` the branch you want to update on the remote

or

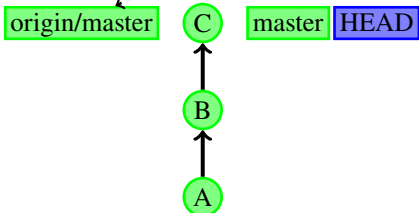


```
git@test: git fetch origin
git@test: git push origin master
```

# Uploading your changes I - no conflicts

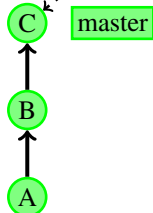
Local Repository

the remote branch  
is updated



Remote Repository

your commit is now  
known to the server

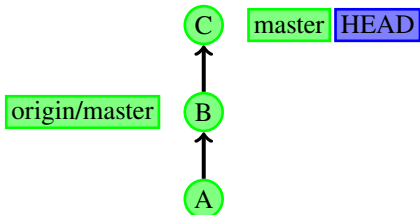


```
git@test: git fetch origin
git@test: git push origin master
```



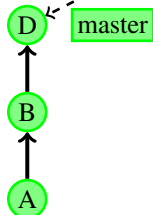
# Uploading your changes II - trivial conflicts

Local Repository



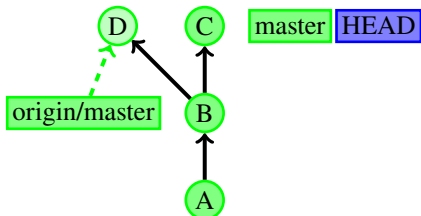
Remote Repository

scenario II: a new commit with trivial conflicts exists

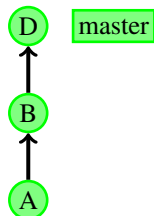


# Uploading your changes II - trivial conflicts

Local Repository



Remote Repository

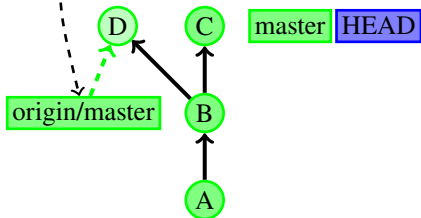


```
git@test: git fetch origin
```

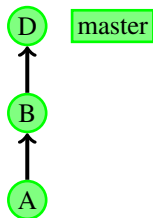
# Uploading your changes II - trivial conflicts

Local Repository

fetch only updates  
the remote branch



Remote Repository

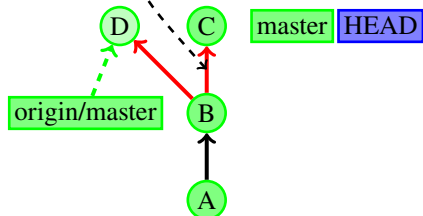


```
git@test: git fetch origin
```

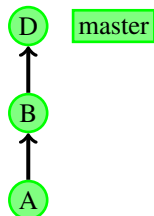
# Uploading your changes II - trivial conflicts

Local Repository

this fork in the history  
must be resolved



Remote Repository



```
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

origin/master

B



B

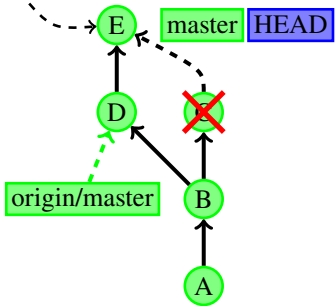


```
git@test: git fetch origin
git@test: git rebase origin/master
```

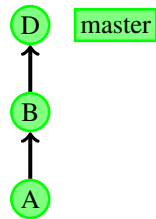
# Uploading your changes II - trivial conflicts

Local Repository

"E" contains the same changes as the deleted "C"



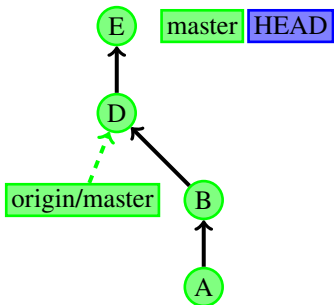
Remote Repository



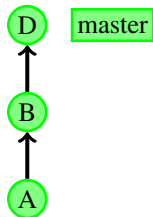
```
git@test: git fetch origin
git@test: git rebase origin/master
```

# Uploading your changes II - trivial conflicts

Local Repository



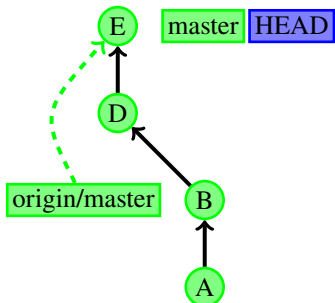
Remote Repository



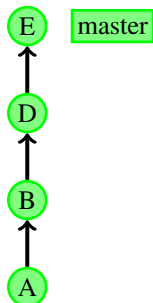
```
git@test: git fetch origin
git@test: git rebase origin/master
```

# Uploading your changes II - trivial conflicts

Local Repository



Remote Repository

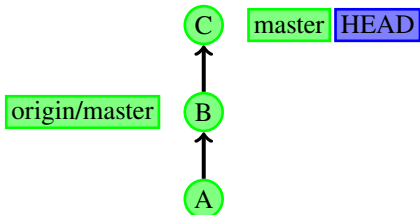


```
git@test: git fetch origin
git@test: git rebase origin/master
git@test: git push origin master
```



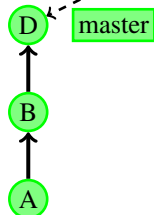
# Uploading your changes III - real conflicts

Local Repository



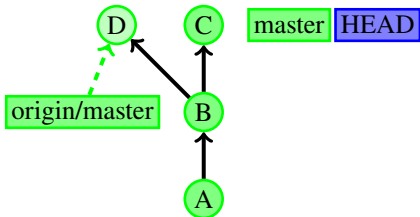
Remote Repository

scenario III: a new commit with real conflicts exists

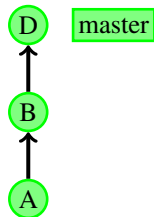


# Uploading your changes III - real conflicts

Local Repository



Remote Repository

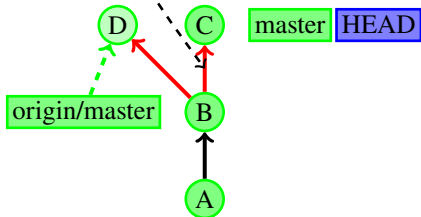


```
git@test: git fetch origin
```

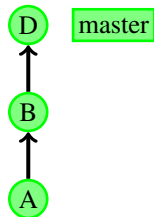
# Uploading your changes III - real conflicts

Local Repository

conflicting changes  
must be resolved



Remote Repository



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

# Uploading your changes III - real conflicts

Local Repository

Remote Repository



```
git merge [--ff-only] <branch>  
merge changes from another branch into the current branch  
branch the branch you want to merge with the current one  
--ff-only allow fast-forward merge only, useful for updating  
unchanged local branches after fetching from a remote
```

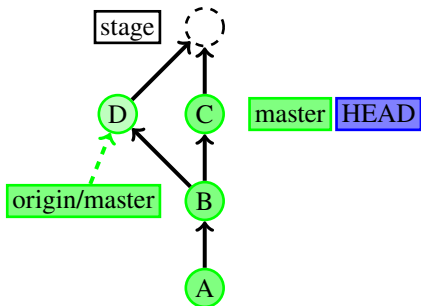
or



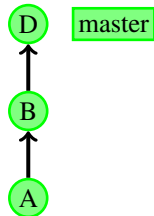
```
git@test: git fetch origin  
git@test: git merge origin/master
```

# Uploading your changes III - real conflicts

Local Repository



Remote Repository

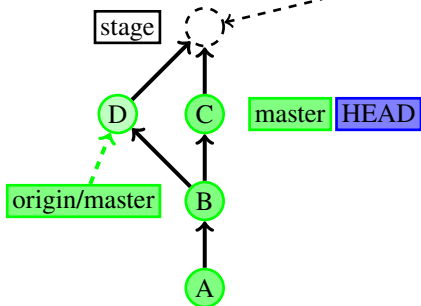


```
git@test: git fetch origin
git@test: git merge origin/master
```

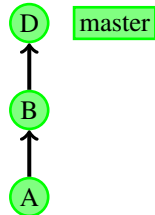
# Uploading your changes III - real conflicts

Local Repository

no commit made, have to resolve conflict



Remote Repository



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

# Uploading your changes III - real conflicts

Local Repository

Remote Repository

```
git mergetool [--tool=...] <path>
```

use the tool of your choice to manually resolve conflicts  
for 3-way diff-tools, the BASE version will be staged after resolution

`path` the file(s) to resolve

`--tool` specify the tool you want to use, e.g. meld or vimdiff

or

A

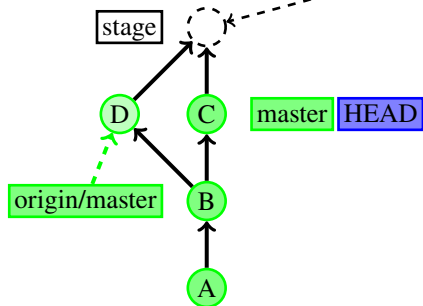
A

```
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

all conflicts resolved,  
time to commit



Remote Repository

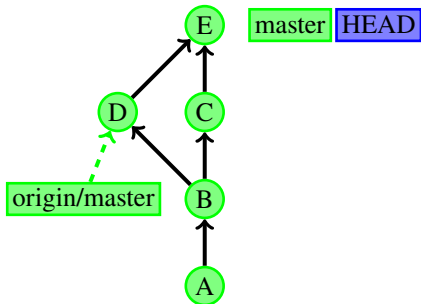


```
git@test: git fetch origin
git@test: git merge origin/master
git@test: git mergetool nasty_conflict.py
```



# Uploading your changes III - real conflicts

Local Repository



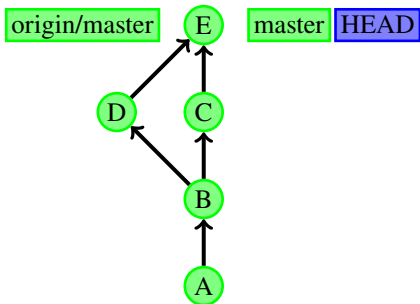
Remote Repository



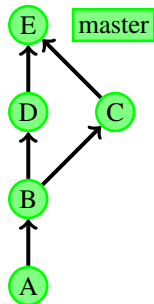
```
git@test: git fetch origin
git@test: git merge origin/master
git@test: git mergetool nasty_conflict.py
git@test: git commit
```

# 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
```

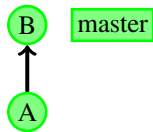
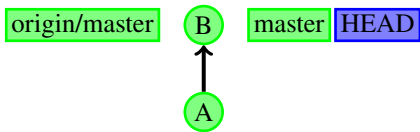
# git branches

## organizing and isolating your development

# Committing your changes into a new local branch

Local Repository

Remote Repository



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

# Committing your changes into a new local branch

Local Repository

Remote Repository

```
git branch [-a] [<name> [<start>]]
```

list all known branches (if no argument is given) or  
create a new local branch originating at the chosen starting-point

**name** a unique label for the new branch

**start** the starting point of the branch, defaults to HEAD

or

**-a** list all branches, both local and remote

A

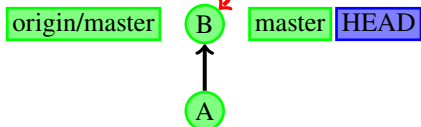
A

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

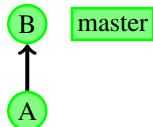
# Committing your changes into a new local branch

Local Repository

the new branch points  
to your current HEAD



Remote Repository



```
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

origin/master

B

master

HEAD



B

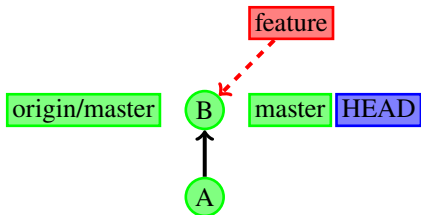
master



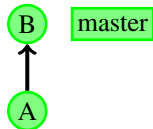
```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
```

# Committing your changes into a new local branch

Local Repository



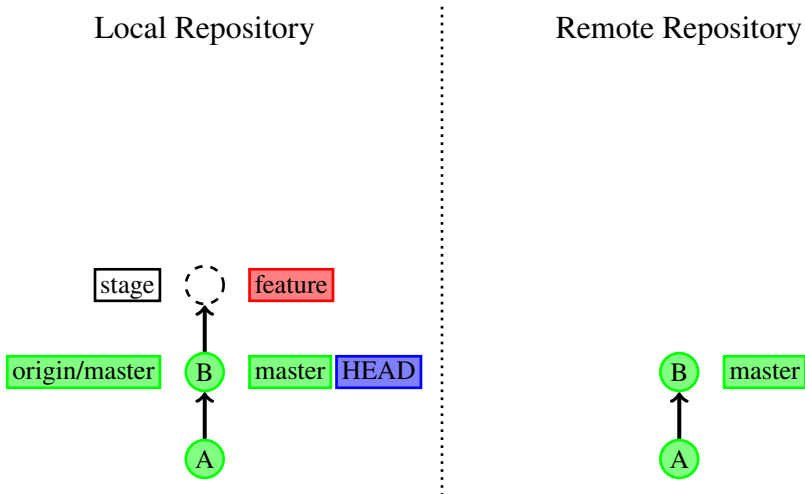
Remote Repository



```
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

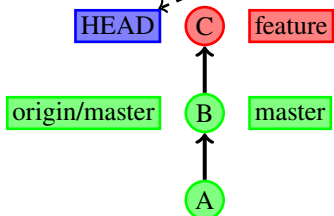


```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
git@test: git add newfile.txt
```

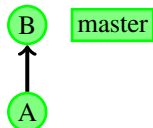
# Committing your changes into a new local branch

Local Repository

HEAD moved to the new working copy state



Remote Repository



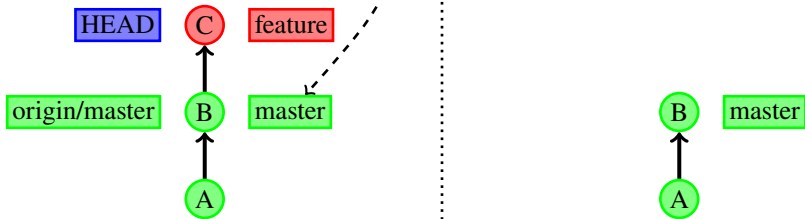
```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
git@test: git add newfile.txt
git@test: git commit
```

# Committing your changes into a new local branch

Local Repository

Remote Repository

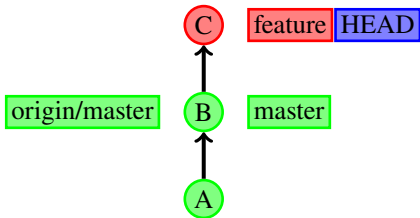
the master branch  
is unchanged



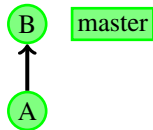
```
git@test: echo "This should be fun!" > newfile.txt
git@test: git branch feature
git@test: git checkout feature
git@test: git add newfile.txt
git@test: git commit
```

# Uploading your changes IV - a new branch

Local Repository



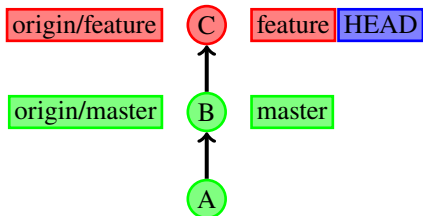
Remote Repository



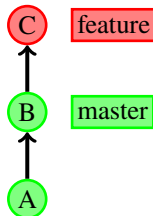
```
git@test: git fetch origin
```

# Uploading your changes IV - a new branch

Local Repository



Remote Repository

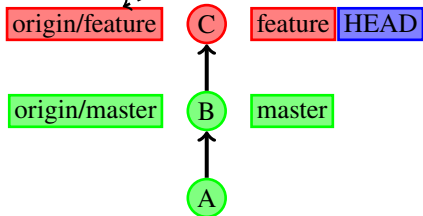


```
git@test: git fetch origin
git@test: git push origin feature
```

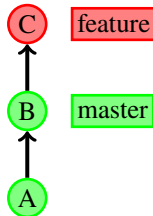
# Uploading your changes IV - a new branch

Local Repository

a new remote branch  
has been created



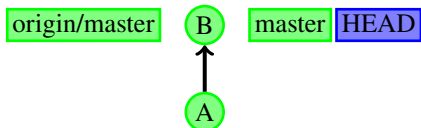
Remote Repository



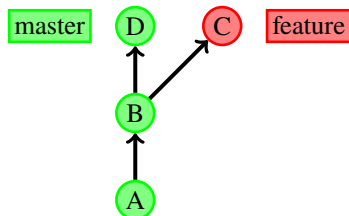
```
git@test: git fetch origin
git@test: git push origin feature
```

# Working with branches - turning remote into local branches

Local Repository

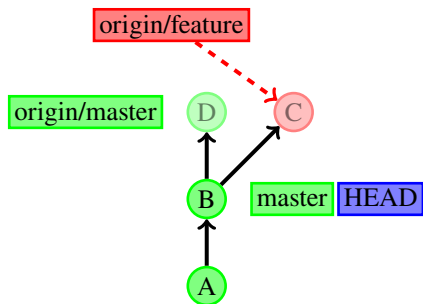


Remote Repository

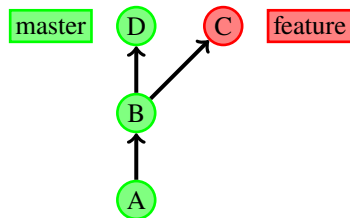


# Working with branches - turning remote into local branches

Local Repository



Remote Repository



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



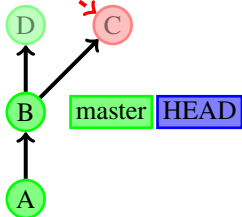
# Working with branches - turning remote into local branches

Local Repository

a new remote branch  
has been created

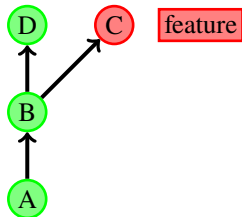
origin/feature

origin/master



Remote Repository

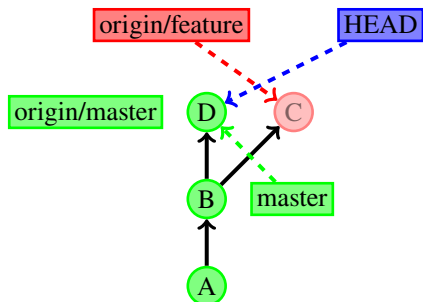
master



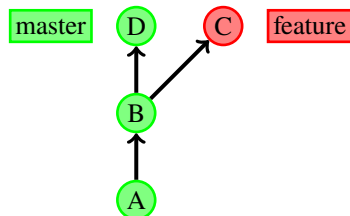
```
git@test: git fetch origin
```

# Working with branches - turning remote into local branches

Local Repository

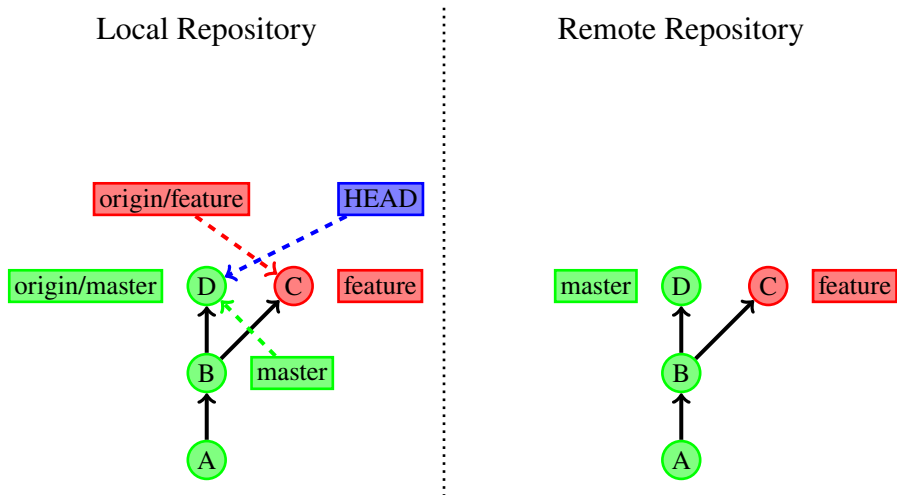


Remote Repository



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

# Working with branches - turning remote into local branches

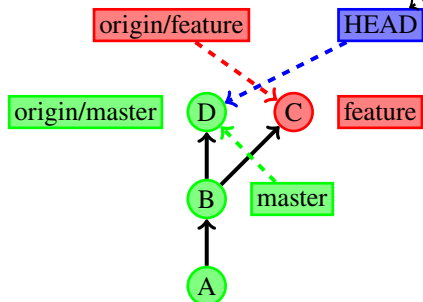


```
git@test: git fetch origin
git@test: git merge --ff-only origin/master
git@test: git branch feature origin/feature
```

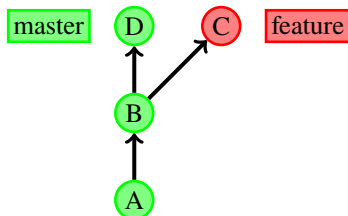
# Working with branches - turning remote into local branches

Local Repository

HEAD has not moved!  
working copy unchanged

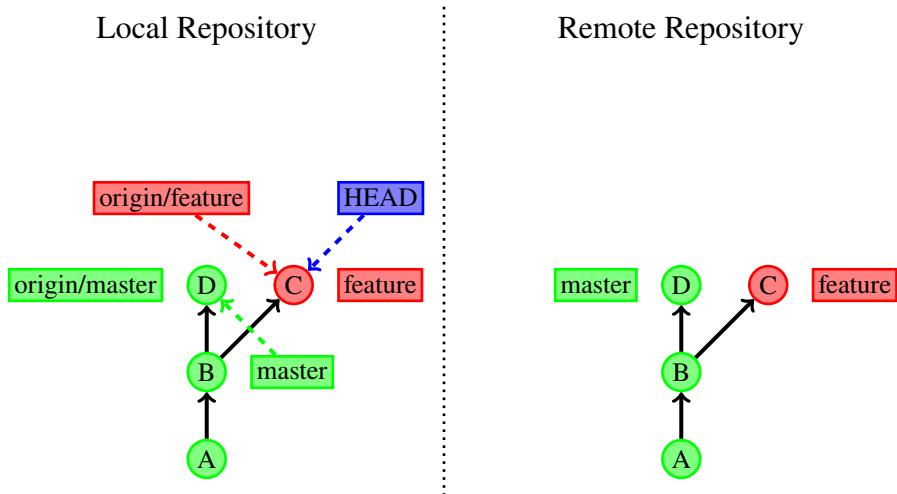


Remote Repository



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

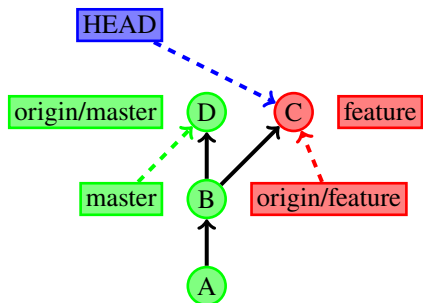
# Working with branches - turning remote into local branches



```
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
```

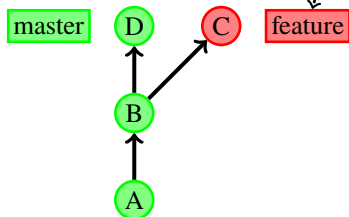
# Working with branches I - merging finished branches

Local Repository



Remote Repository

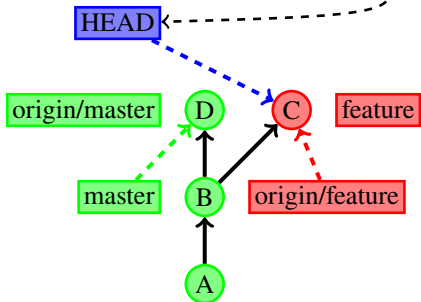
Scenario I:  
feature ready to merge



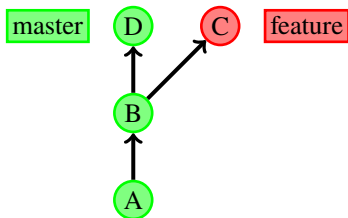
# Working with branches I - merging finished branches

Local Repository

move HEAD to master  
to merge *into* master

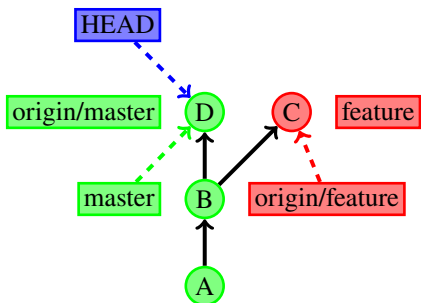


Remote Repository

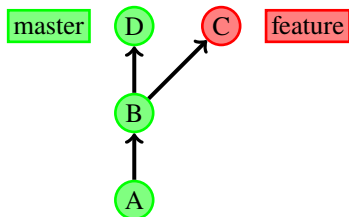


# Working with branches I - merging finished branches

Local Repository



Remote Repository

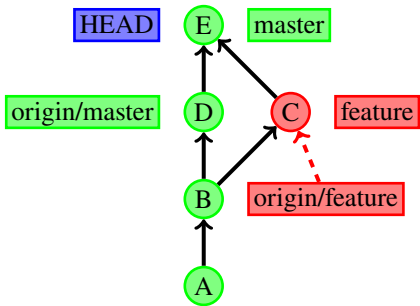


```
git@test: git checkout master
```

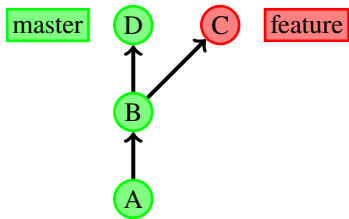


# Working with branches I - merging finished branches

Local Repository



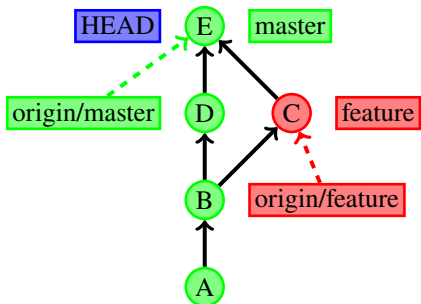
Remote Repository



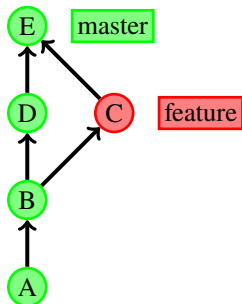
```
git@test: git checkout master  
git@test: git merge feature
```

# Working with branches I - merging finished branches

Local Repository



Remote Repository

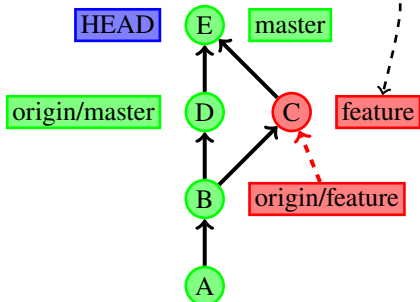


```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
```

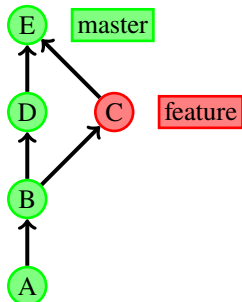
# Working with branches I - merging finished branches

Local Repository

feature branch is no longer used, remove it



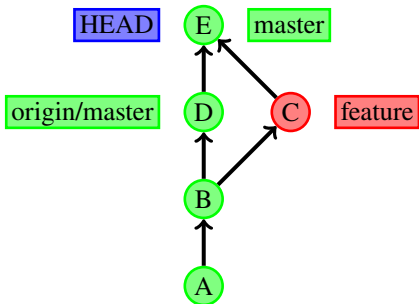
Remote Repository



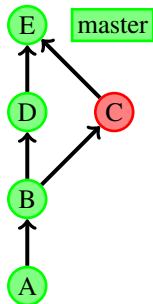
```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
```

# Working with branches I - merging finished branches

Local Repository



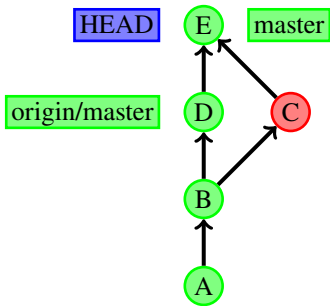
Remote Repository



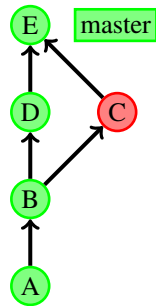
```
git@test: git checkout master
git@test: git merge feature
git@test: git push origin master
git@test: git push --delete origin feature
```

# Working with branches I - merging finished branches

Local Repository



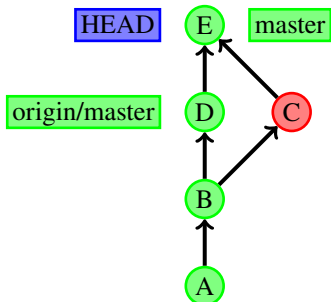
Remote Repository



```
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
```

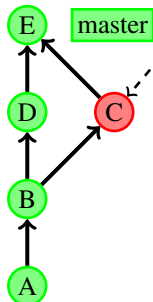
# Working with branches I - merging finished branches

Local Repository



Remote Repository

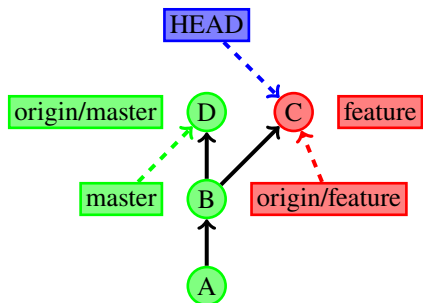
removing the branch  
causes no history loss



```
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
```

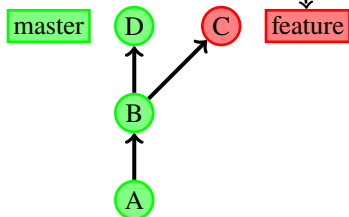
# Working with branches II - updating feature branches

Local Repository



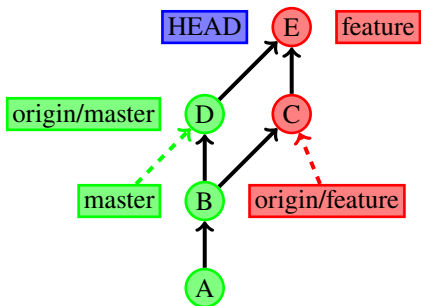
Remote Repository

Scenario II: feature needs new content from master branch

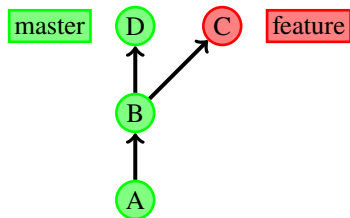


# Working with branches II - updating feature branches

Local Repository



Remote Repository



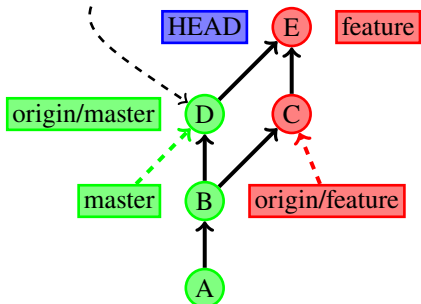
```
git@test: git merge master
```



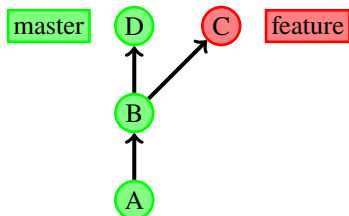
# Working with branches II - updating feature branches

Local Repository

master is unchanged



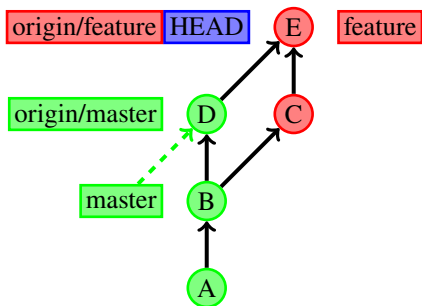
Remote Repository



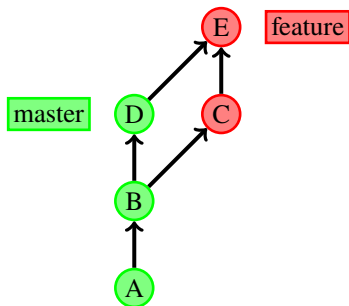
```
git@test: git merge master
```

# 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

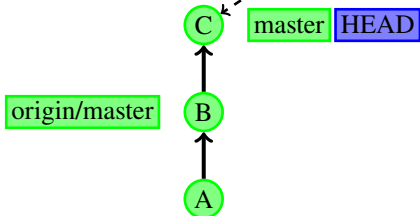
**revision** the revision you want to restore, usually HEAD

**path** shell pattern for the files to revert

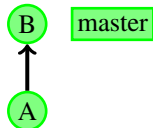
# Undoing commits - deleting a local only commit

Local Repository

Scenario I: this local commit is faulty



Remote Repository



# 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

or

A

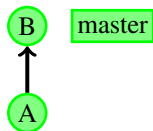
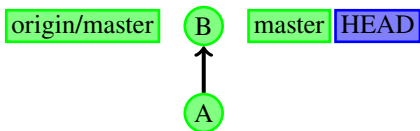
A

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

# Undoing commits - deleting a local only commit

Local Repository

Remote Repository



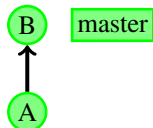
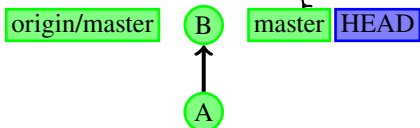
```
git@test: git reset --hard HEAD~1
```

# Undoing commits - deleting a local only commit

Local Repository

Remote Repository

git reset HEAD~1 moved  
back the tip by 1 commit



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

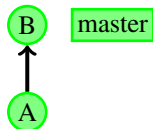
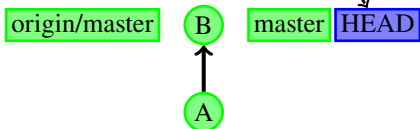


# Undoing commits - deleting a local only commit

Local Repository

Remote Repository

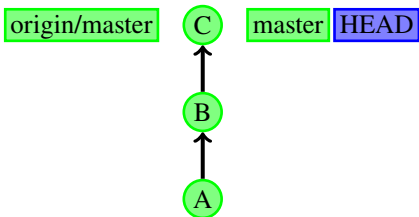
`--hard` resets all files  
to their state in this commit



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

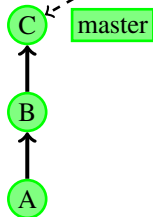
# Undoing commits - reverting an already pushed commit

Local Repository



Remote Repository

Scenario II: this already pushed commit is faulty



# Undoing commits - reverting an already pushed commit

Local Repository

Remote Repository

```
git revert <revision>
```

create a new commit which reverts the  
changes introduced by the specified commit

`revision` the revision you want to undo

Note: take special care when reverting merges!  
(see `git help revert`)

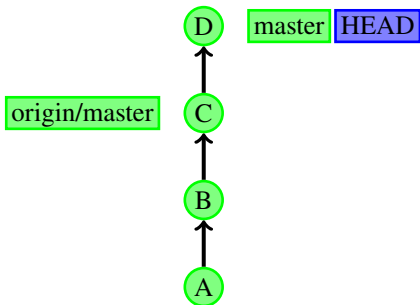
A

A

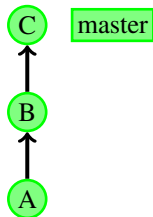
```
git@test: git revert HEAD
```

# Undoing commits - reverting an already pushed commit

Local Repository



Remote Repository

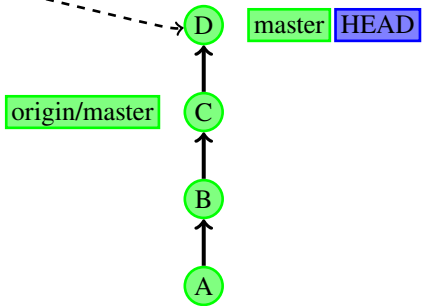


```
git@test: git revert HEAD
```

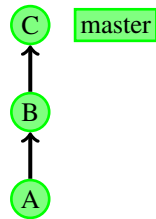
# Undoing commits - reverting an already pushed commit

Local Repository

this new commit undoes  
all changes from C



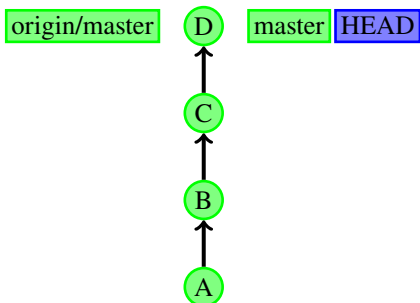
Remote Repository



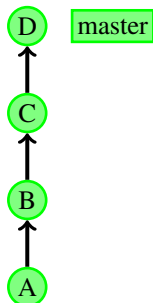
```
git@test: git revert HEAD
```

# 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!"
```