





Stefan Rotman
rotman@puzzle.ch

1. Introducing Git
2. Centralized vs Distributed VCS
3. Git Basics
4. Git Branches
5. Git Distributed
6. Workflow Strategies

Agenda

\$ whoami

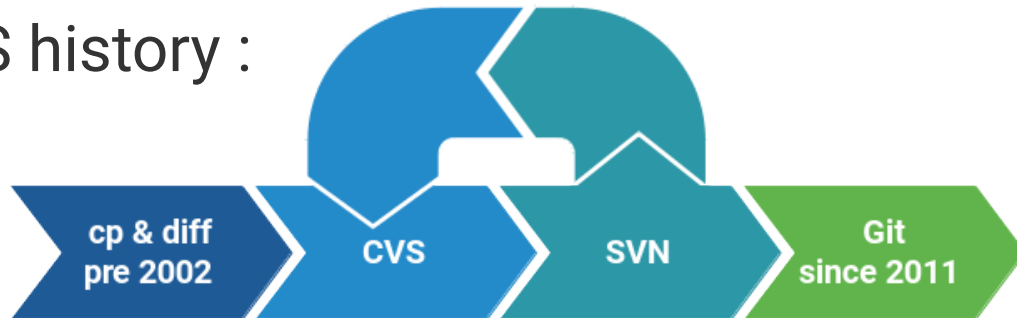
Stefan Rotman <rotman@puzzle.ch>

Puzzle ITC – since 2010 [Software Engineer, Software Architect]

Java Enthusiast – since 1997

Git Enthusiast – since 2011

My personal VCS history :



The top corners of the slide feature decorative geometric shapes. In the top-left corner, there are overlapping triangles in shades of blue and teal. In the top-right corner, there are overlapping squares and triangles in shades of blue and teal.

1

Introducing Git

History of Git

Developed in 2005

as VCS for the Linux kernel

by Junio Hamano and Linus Torvalds

Fundamental requirements

Performance and distributed nature

Security and trust issues

Why the name ?

... pick and choose :

- *« I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'Git' » – Linus Torvalds*
- *Random three-letter combination not yet used on UNIX*
- *Slang meanings*
- *...*

What is Git ?

Git is a free and open source distributed version control system designed for speed and efficiency.

What is Git ?

Git is a free and open source **distributed** version control system **designed for speed and efficiency**.

Git is fully distributed

(almost) everything happens local

it's fast !

every clone is a full repository backup

... and a potential git server

it works offline

(almost) everything happens local

No network access needed to

- commit changes

- generate diffs

- inspect commit history

- create branches

- merge branches

- change between branches

- ... and much more 😊

Security, trust and efficiency

Every commit is immutable

Every commit is a snapshot of the entire tree, not just a patch

Data integrity is central

Branching & merging is cheap, fast and easy!

Git rarely removes data (it just removes it from the working tree)

Commits are immutable

Every commit is identified by a unique SHA-1 code

Every commit points to a snapshot of the entire tree

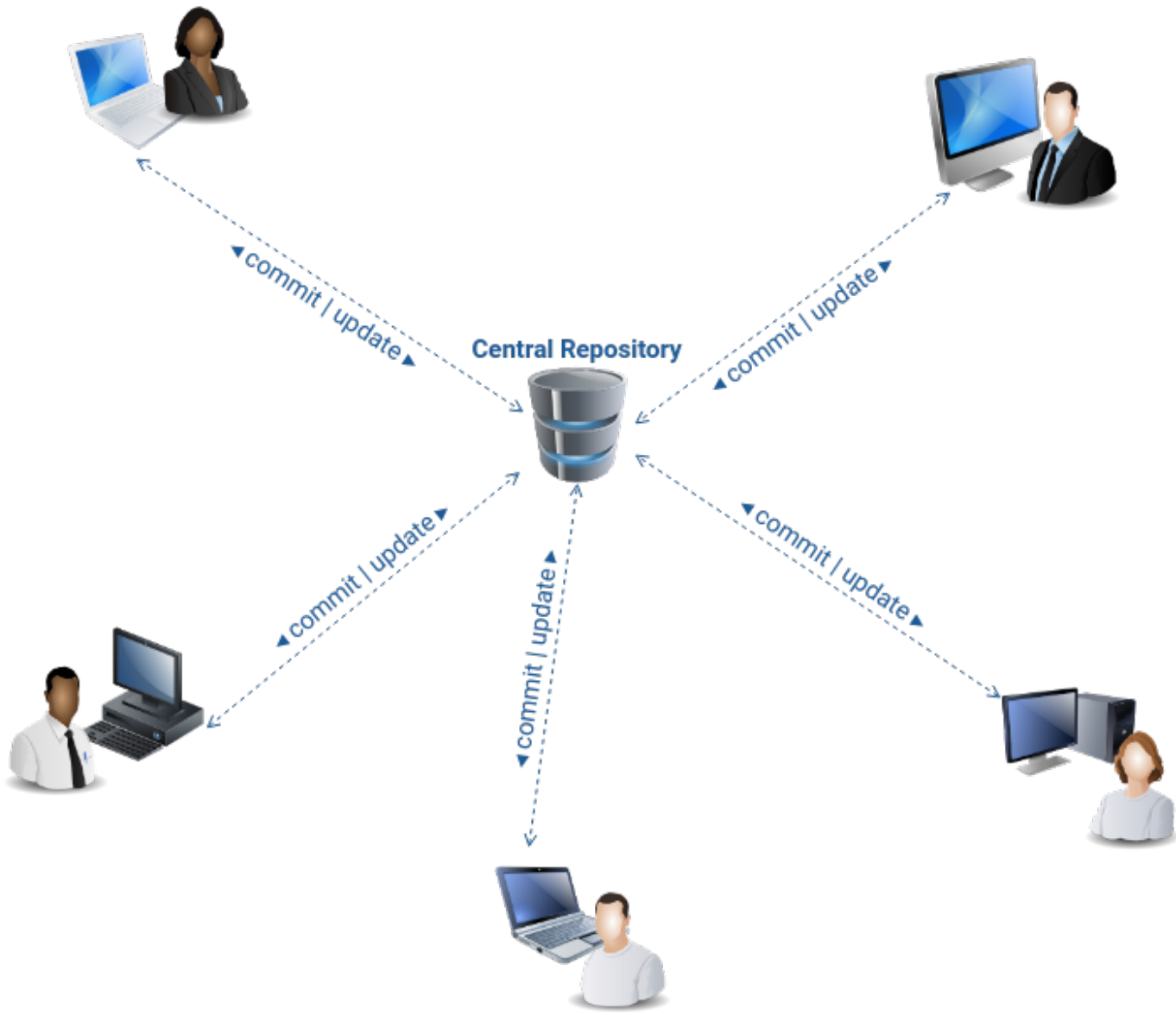
7088c2eeb65c6471f91263873db7ebef63f039d7	
tree	43e184c8bb464a6ee1a6538e4fda0dc334aa010f
parent	aa2cf3e503aee6824aafc226d805cf8a722ce70a
author	Stefan Rotman <rotman@puzzle.ch> 1466004627 +0200
committer	Stefan Rotman <rotman@puzzle.ch> 1466004806 +0200
A Commit message	



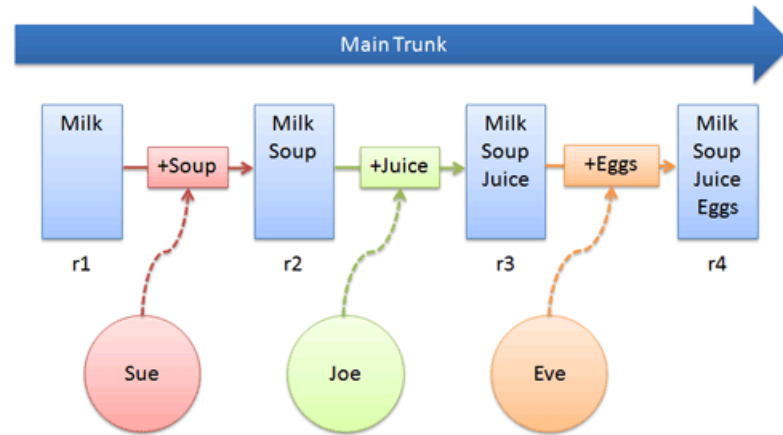
2

Centralized VCS vs Distributed VCS

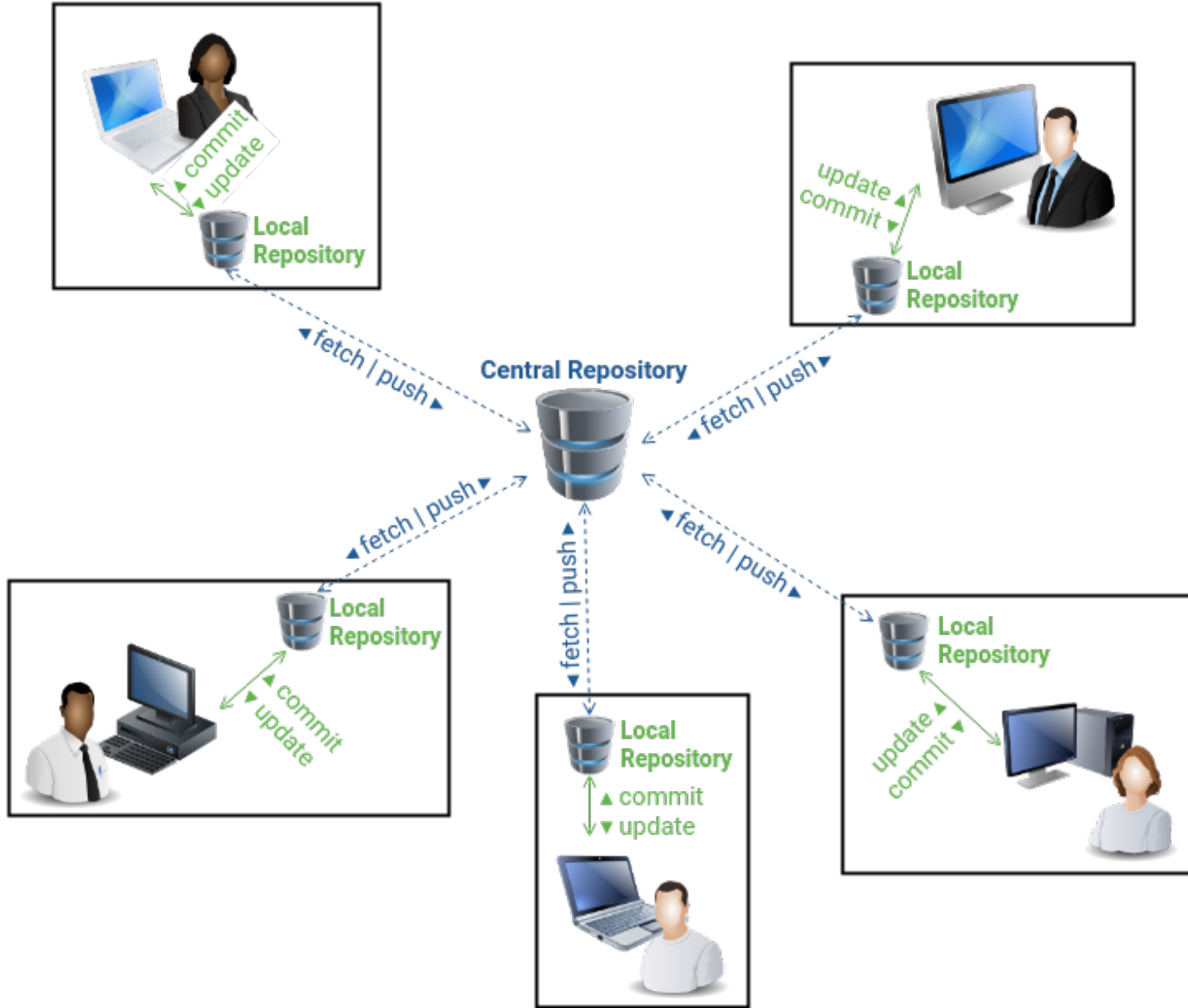
Centralized VCS



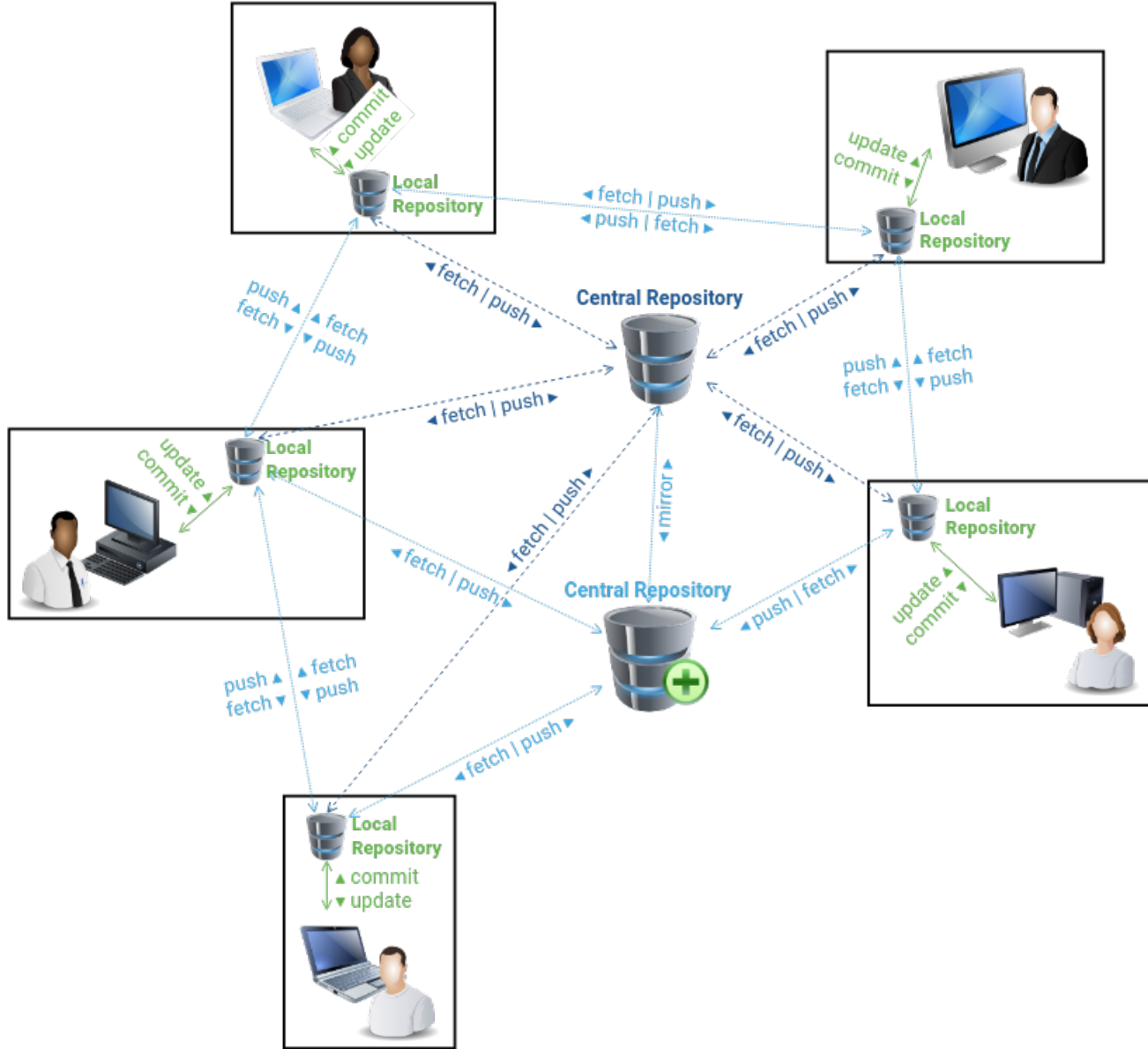
Centralized VCS



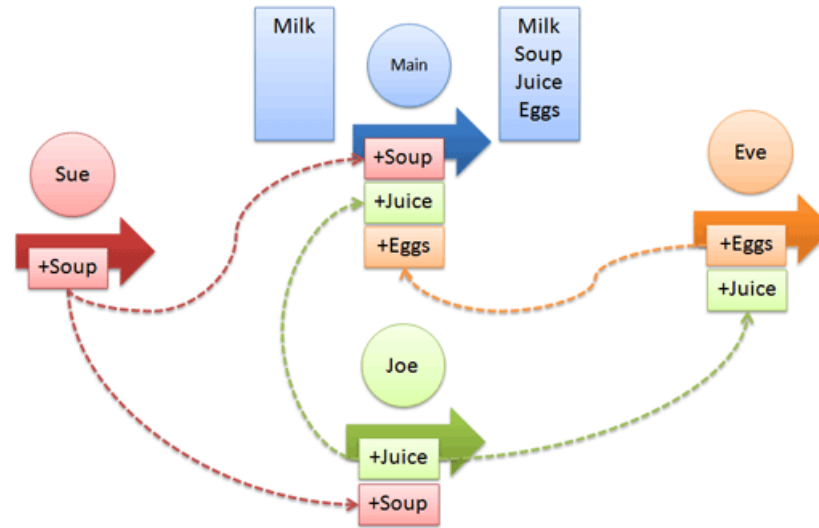
Distributed VCS



Distributed VCS



Distributed VCS



3

Git Basics

State of the data

In 'classic' VCS, data can live in 2 places:

In the datastore → for Git, that is local repository
 → HEAD references the last commit

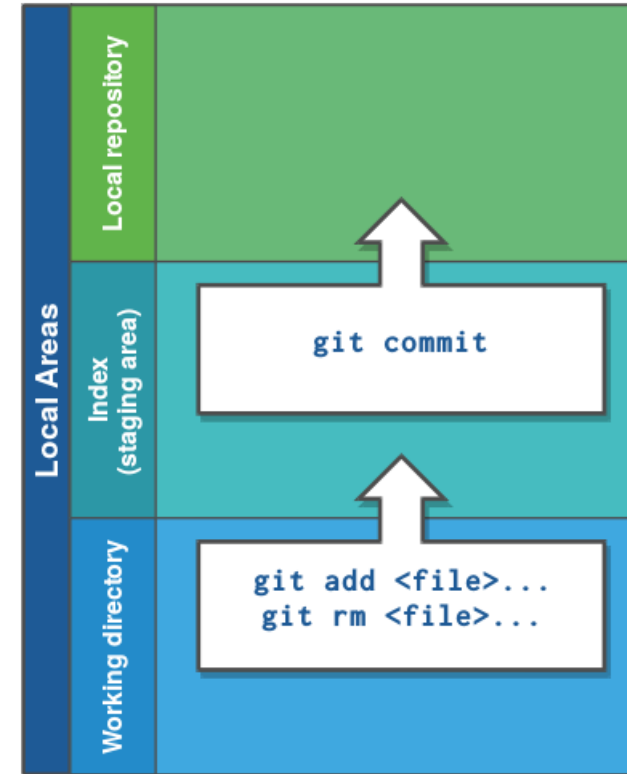
In the working Directory

Git adds an area between the working directory and the datastore:

The index

From change to commit

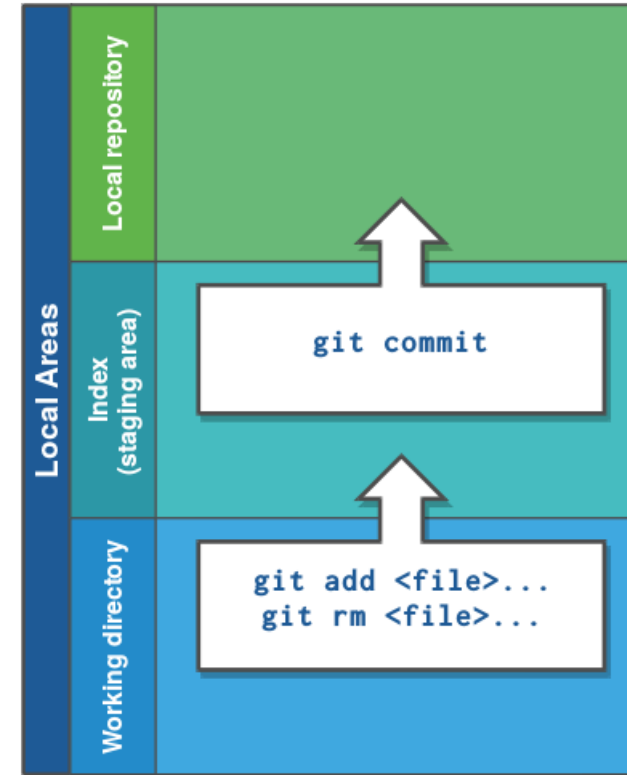
Changes are made in the Working directory



From change to commit

Changes are staged in the Index

Changes are made in the Working directory

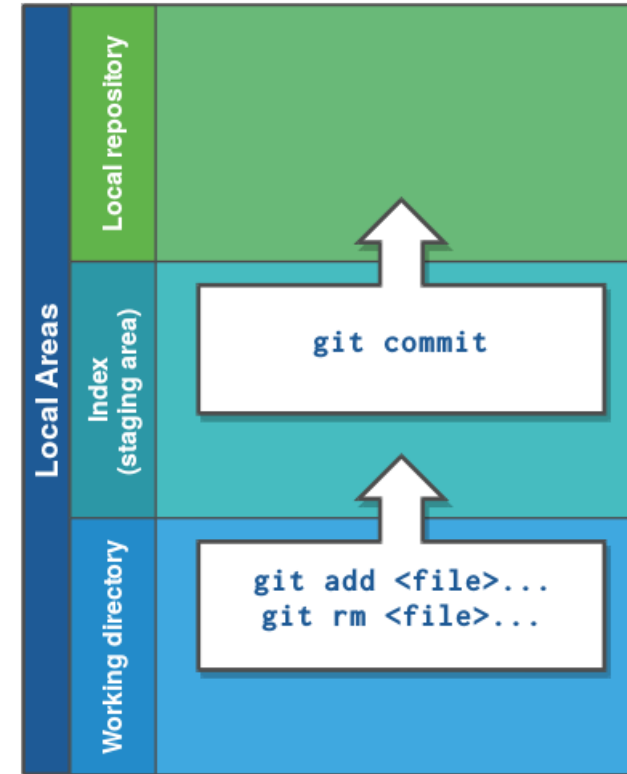


From change to commit

Changes are committed to the local repository
this updates the HEAD reference

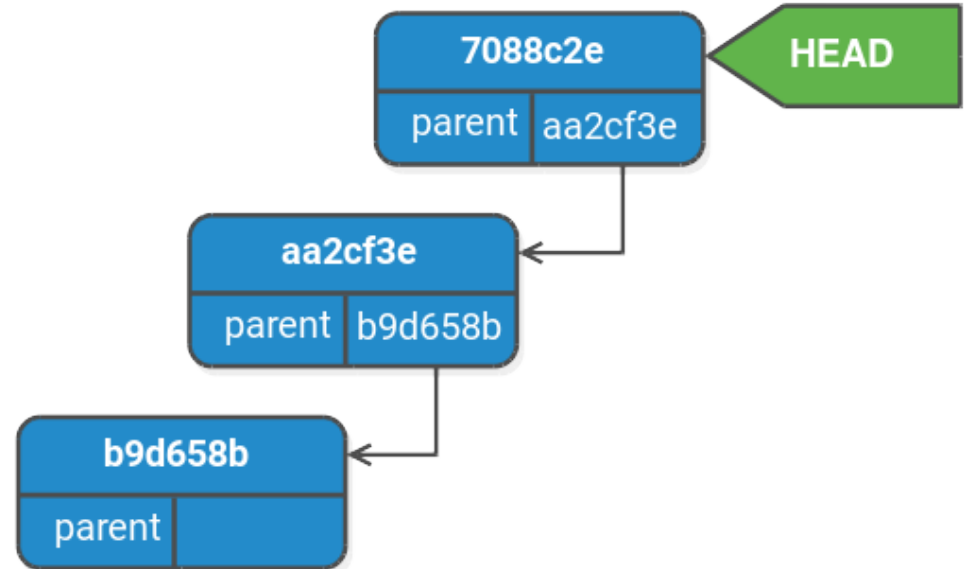
Changes are staged in the Index

Changes are made in the Working directory



Local Repository: the HEAD tree

7088c2eeb65c6471f91263873db7ebef63f039d7	
tree	43e184c8bb464a6ee1a6538e4fda0dc334aa010f
parent	aa2cf3e503aee6824aafc226d805cf8a722ce70a
author	Stefan Rotman <rotman@puzzle.ch> 1466004627 +0200
committer	Stefan Rotman <rotman@puzzle.ch> 1466004806 +0200
A Commit message	



Making yourself known

git config is used to configure git repositories

with **user.name** and **user.email** values identify a user

The **--global** flag makes the setting a user default

```
$ git config --global user.name "Stefan Rotman"  
$ git config --global user.email rotman@puzzle.ch  
$ git config -l | grep user.  
user.name Stefan Rotman  
user.email rotman@puzzle.ch
```

Creating a clean repository

git init initializes a new repository

No files are added yet, no commits are made yet



*The 'administration' for a git repository is located in the **.git** folder*

```
$ mkdir foo
$ cd foo
$ git init
Initialized empty Git repository in
/home/srotman/foo/.git/
$ ls -A
.git
```

Cloning a remote repository

git clone clones a remote repository into a local repository



*The default name git uses for a remote repository is **origin***

```
$ git clone git://foo.com/bar.git
Cloning into 'bar'...
remote: Counting objects: 13, done.
remote: Total 13 (delta 0), reused 0
(delta 0), pack-reused 13
Receiving objects: 100% (13/13), done.
Receiving deltas: 100% (2/2), done.
Checking connectivity... done.

$ cd bar

$ ls -A
.git src README
```

Check for changes

git status checks for uncommitted changes in the working directory and index

```
$ git status
On branch master
nothing to commit, working directory clean

$ echo 'new file' > file.txt

$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Staging changes

git add <file>... adds files or changes and stage in the index

git rm <file>... removes files and stage the removal in the index

*alternatively, you can also use **git add -p <file>...***

```
$ git add README file.txt
```

```
$ git rm dummy
```

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
    modified:   README
```

```
    deleted:    dummy
```

```
    new file:   file.txt
```

Unstaging changes

git reset HEAD unstages all changes

git reset HEAD <file>... unstages changes made to specific files

```
$ git reset HEAD
```

```
$ git status
```

```
On branch master
```

```
Changes not staged for commit:
```

```
  (use "git add/rm <file>..." to update what will be committed)
```

```
  (use "git checkout -- <file>..." to discard changes in working directory)
```

```
        modified:   README
```

```
        deleted:    dummy
```

```
Untracked files:
```

```
  (use "git add <file>..." to include in what will be committed)
```

Commit staged changes

git commit commits the changes in the index into the local repository

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README
        deleted:    dummy
        new file:   file.txt

$ git commit -m 'Important changes'
[master 3c4bb4d] Important changes
3 files changed, 2 insertions(+), 1 deletion(-)
delete mode 100644 dummy
create mode 100644 file.txt
```


Inspect the commit history

git log shows the commit history of the HEAD

git log <reference> shows the commit history for a specified reference

git log <file> shows the commit history for a specified file

by adding **--follow**, path changes are also considered

git show <commit> shows the changes made in a certain commit



git log --oneline --decorate --graph shows a visual graph of the commit history, including branches, tags, merges, ...

Revert a commit

git revert <commit>

creates a new commit that
undoes the changes from the
specified commit

The commit can be specified by
it's hash, or by a reference to it.

```
$ git revert HEAD --no-edit
[master 4c6fa58] Revert "Important
changes"
 3 files changed, 1 insertion(+), 2
deletions(-)
   create mode 100644 dummy
   delete mode 100644 file.txt

$ git log --oneline -2
4c6fa58 Revert "Important changes"
3c4bb4d Important changes
```

Reset the HEAD

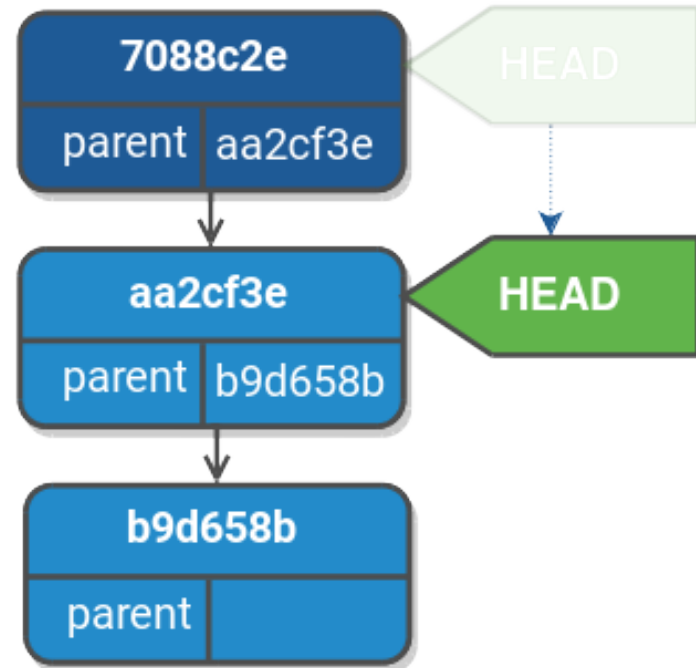
`git reset <reference>`

resets the HEAD to the specified reference (default with `--mixed` behaviour)

--soft loose the commits, but keep the changes in the index

--mixed loose the commits, but keep the changes in the working directory

--hard loose the commits and loose the changes



Undo changes in the working directory

git checkout -- <path>

resets the file (or files) on the specified path to the HEAD version.

If the file is in the index, it will be reset to the version in the Index instead.

The 'time machine'

git checkout <commit>

check out the repository tree as stored under the specified commit. This will land you in a 'detached HEAD' state, meaning that any *new* commit you will make here will not be referenced by a branch or tag

git checkout <commit> <file>

checks out the specified file in the state it has under the specified commit, and places it in the index.

Clean the working directory

git clean

deletes files that are not under Version Control from the working directory.
Usually required one of the following flags:

- i interactive mode. Show what would be done and clean interactively
- f force. Do what needs to be done
- n dry run. Doesn't clean anything, just shows what would be cleaned.

If the **-d** flag is specified, git also cleans up untracked directories.



4

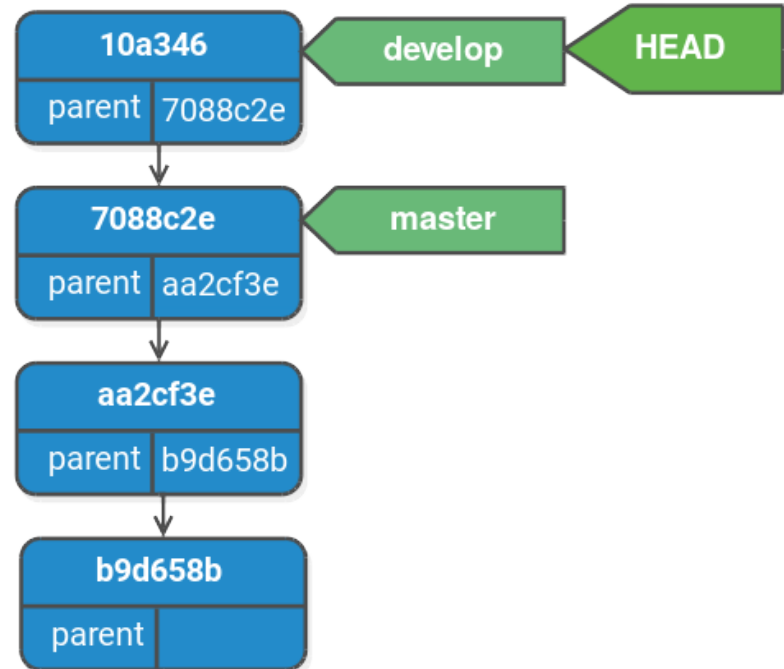
Git Branches

What is a Git Branch

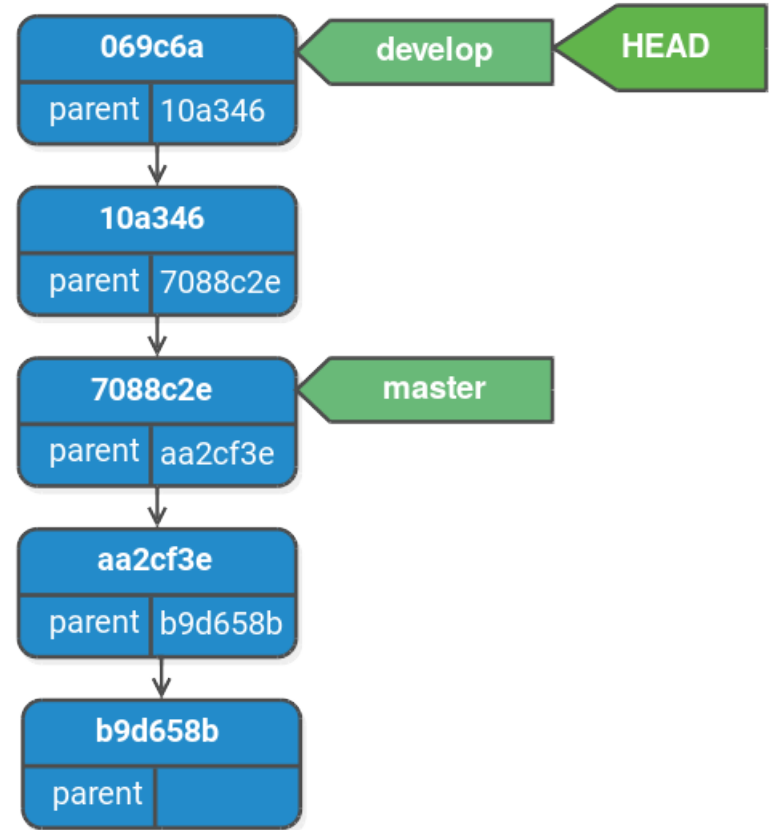
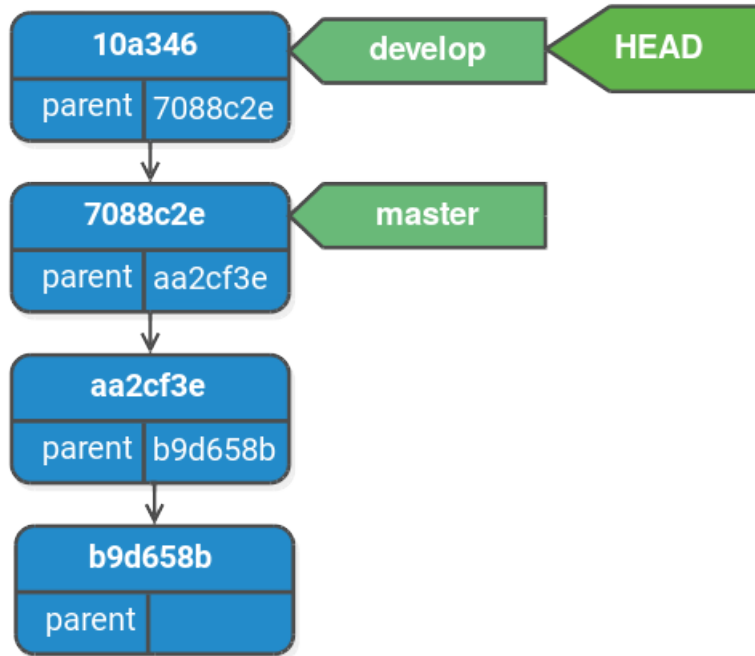
A Branch is a commit marker
(the same is true for tags)

```
$ cat .git/refs/heads/master
7088c2eeb65c6471f91263873db7ebef63f039
d7

$ cat .git/HEAD
ref: refs/heads/develop
```



Commit on active branch



Change between branches

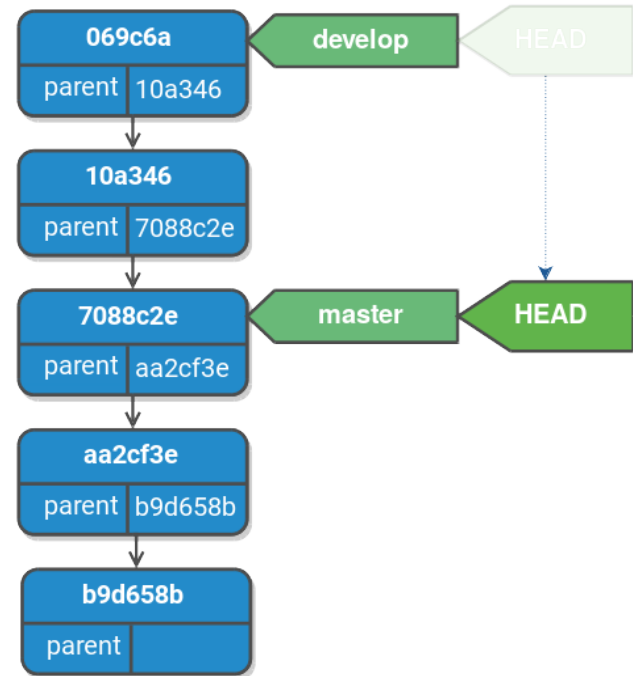
git checkout <branchname>

```
$ git branch
* develop
  master

$ git checkout master
Switched to branch 'master'

$ git branch
  develop
* master

$ cat .git/HEAD
ref: refs/heads/master
```



Create branch

git branch <branchname>

creates a branch on the current HEAD

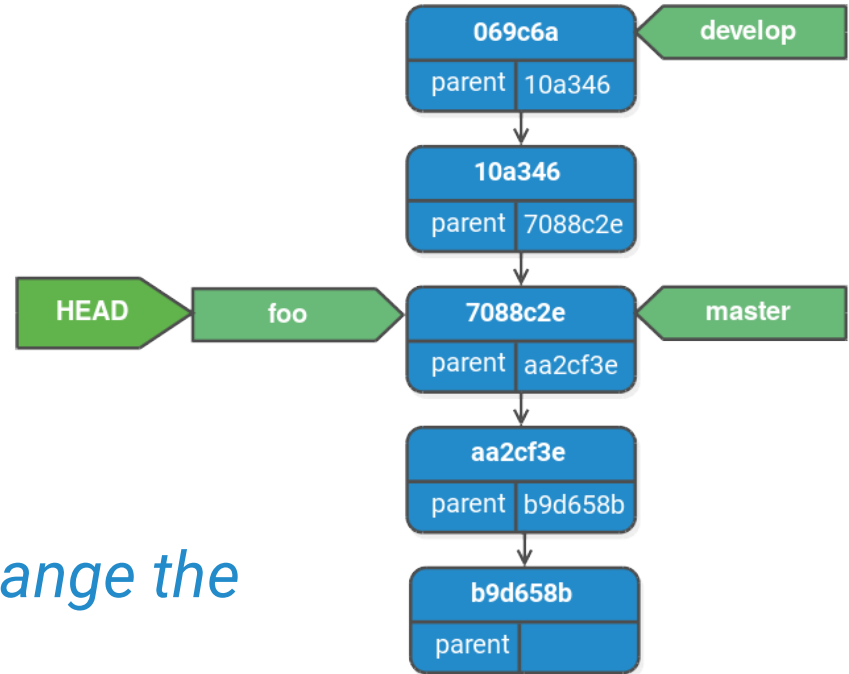
git branch <branchname> <ref>

creates a branch on the specified reference



*Creating a branch does **NOT** change the HEAD reference !*

*Use **git checkout -b <branchname>** to create a branch and check it out right away*



Delete branch

git branch -d will delete a branch that is fully merged in it's upstream or in HEAD

git branch -D will delete a branch without it being merged



You can not delete the branch you're currently on

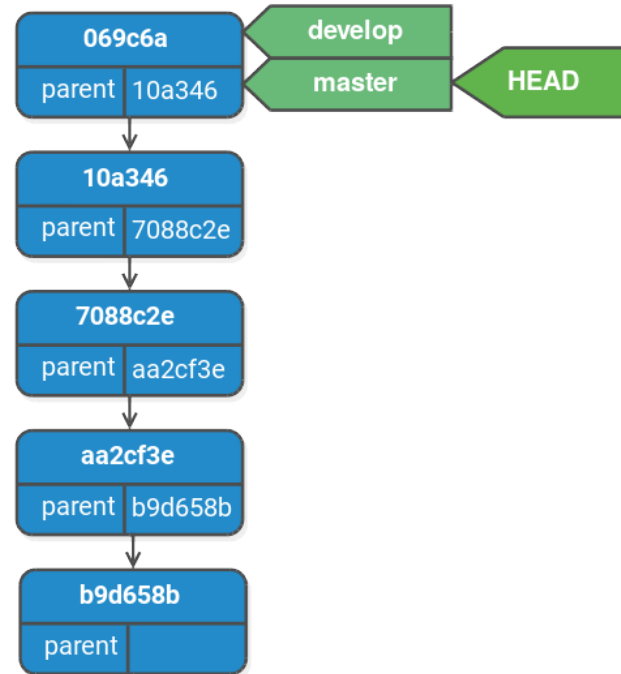
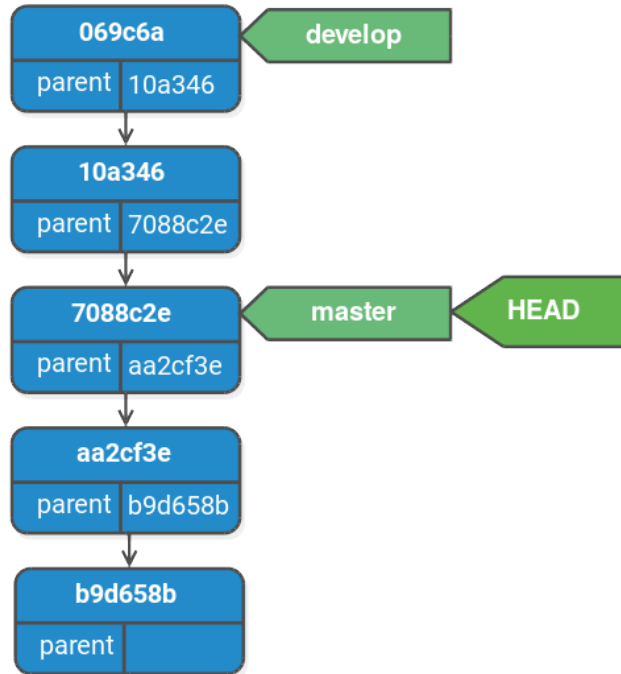
```
$ git branch -D foo
error: Cannot delete the branch 'foo'
which you are currently on.

$ git checkout master
Switched to branch 'master'

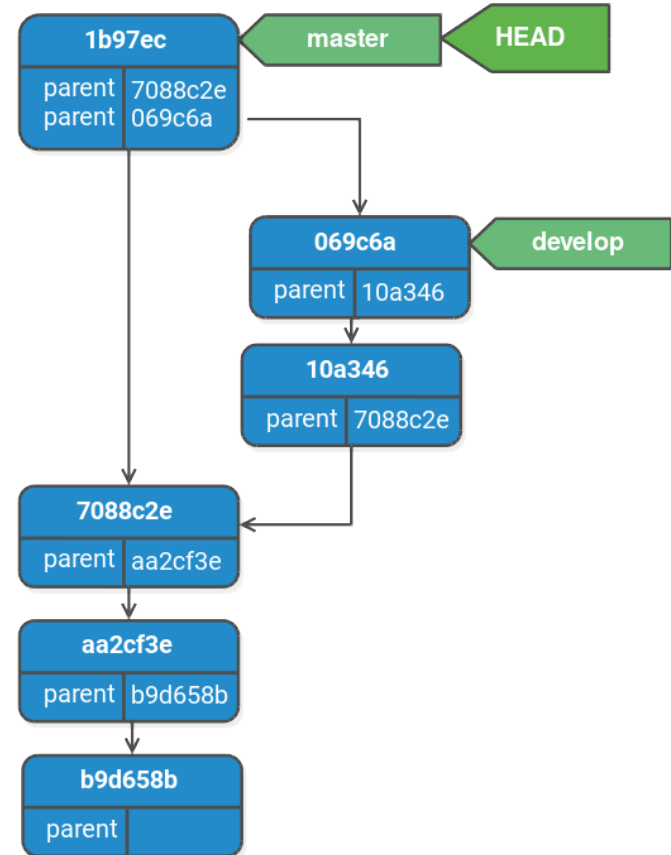
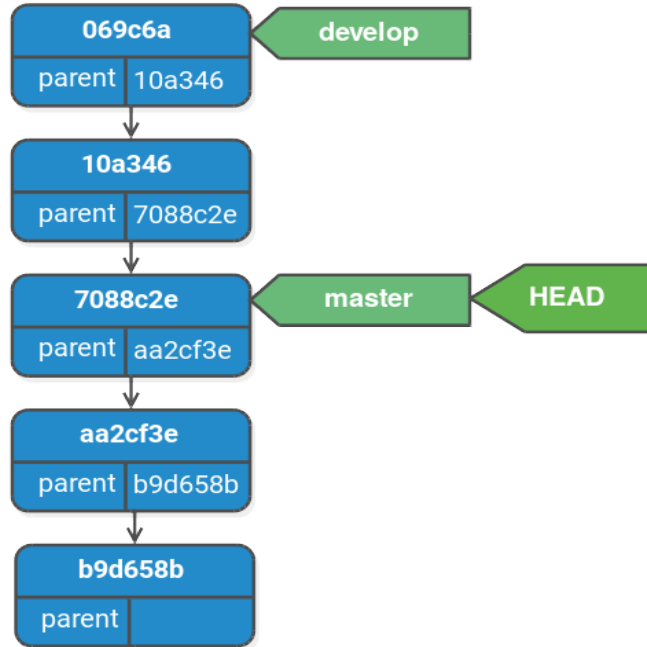
$ git branch -d foo
error: The branch 'foo' is not fully
merged.
If you are sure you want to delete it,
run 'git branch -D foo'.

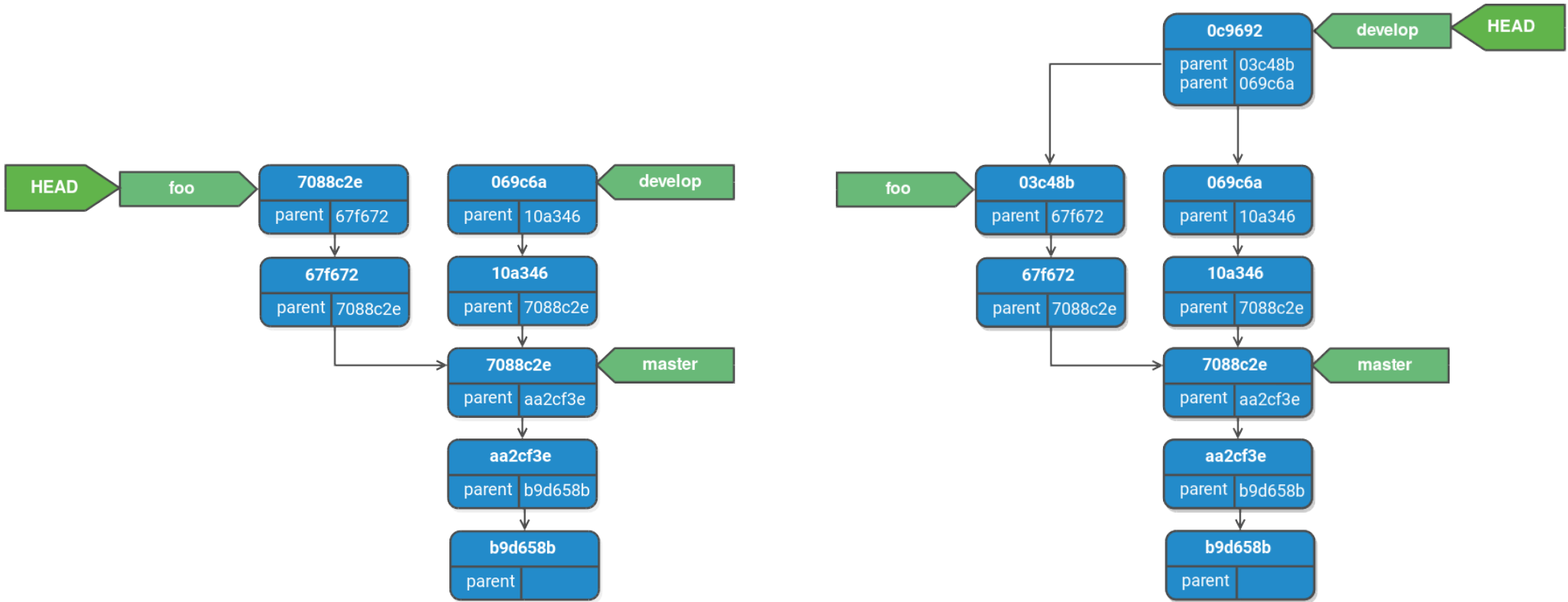
$ git branch -D foo
Deleted branch foo (was e31e45f).
```

Merging (Fast-forward merge)



Merging (non-Fast-forward merge)





Merging branches

git merge <branchname>

merges the specified branch in the active branch.

```
$ git merge develop
Updating 7088c2e..069c6a
Fast-forward
...
```

git merge <to_merge> <target>

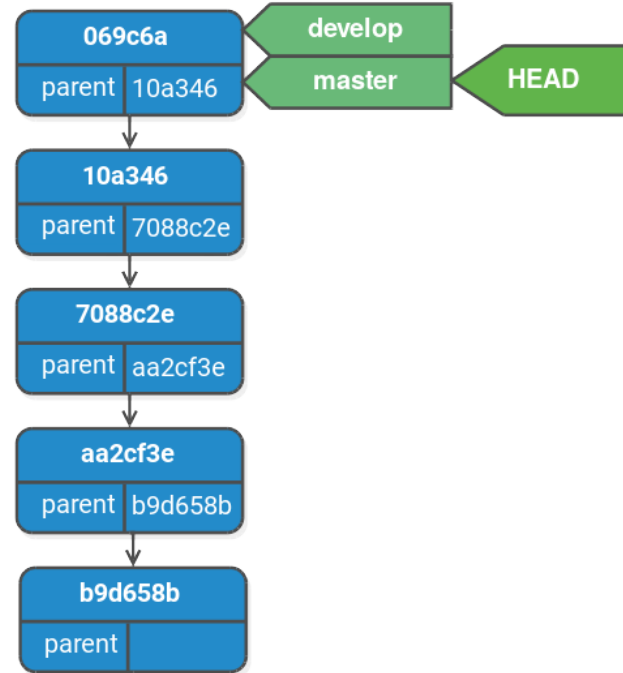
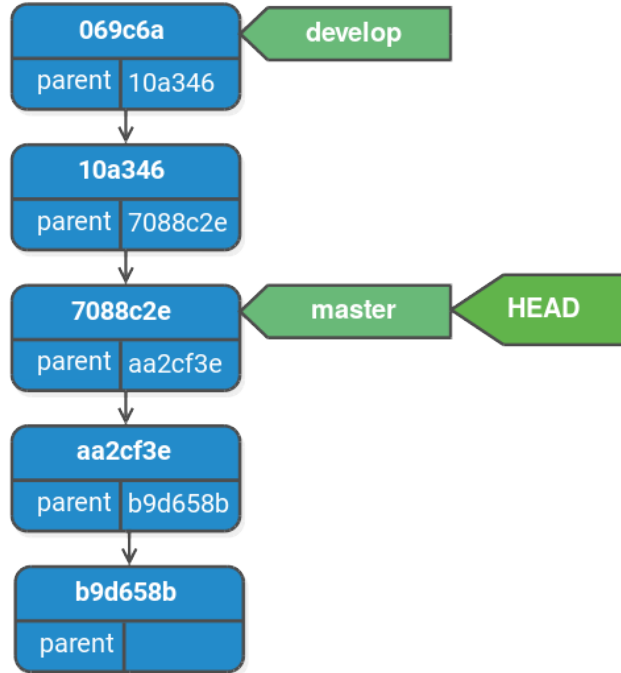
merges the specified branch into the specified target branch.

```
$ git merge --no-ff master
Merge made by the 'recursive'
strategy.
...
```

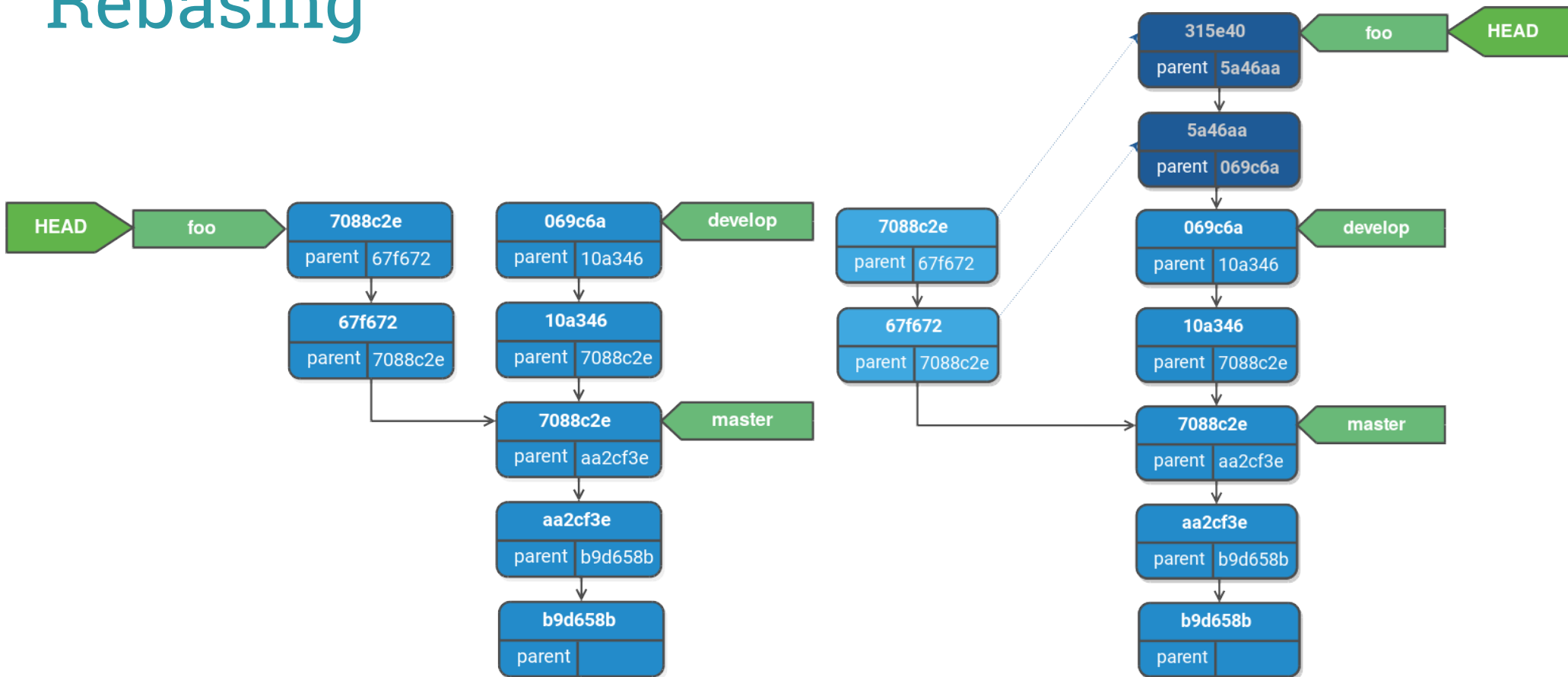


*By default Git will attempt to use Fast-forward merges. Using the **--no-ff** flag enforces a merge commit*

Rebasing (Fast-forward)



Rebasing



Rebasing branches

git rebase <branch>

rebases the active branch onto the specified branch

git rebase <newbase> <branch>

rebases the specified branch onto the specified base

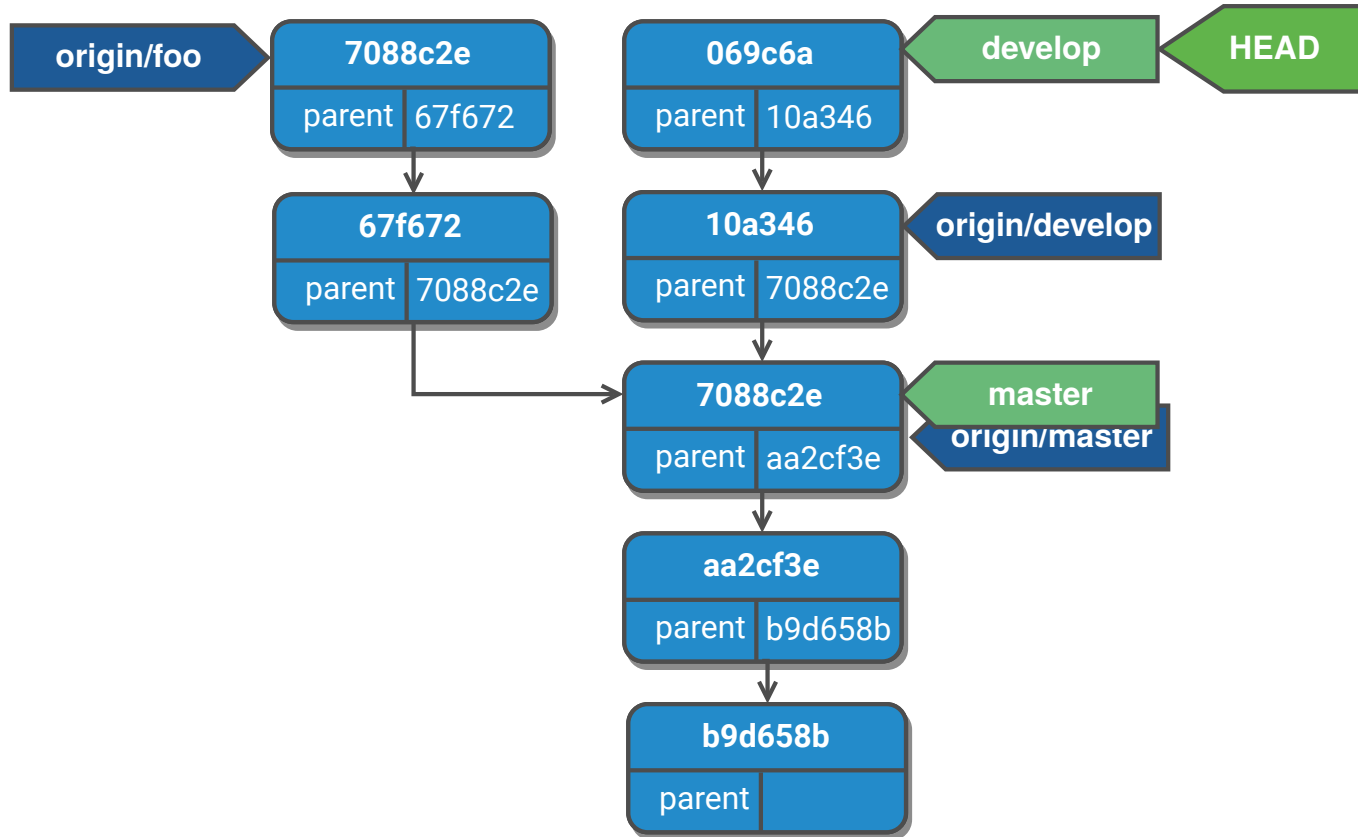
```
$ git checkout master
$ git rebase develop
First, rewinding head to replay your
work on top of it...
Fast-forwarded master to develop.

$ git checkout foo
$ git rebase develop
First, rewinding head to replay your
work on top of it...
Applying: Some Commit message
Applying: Another commit
```

The top corners of the slide feature decorative geometric shapes. In the top-left corner, there are overlapping triangles in shades of light blue, teal, and dark blue. In the top-right corner, there are overlapping squares and triangles in shades of light blue, teal, and dark blue.

4

Git Distributed



Add a remote repository

git remote add <name> <url>

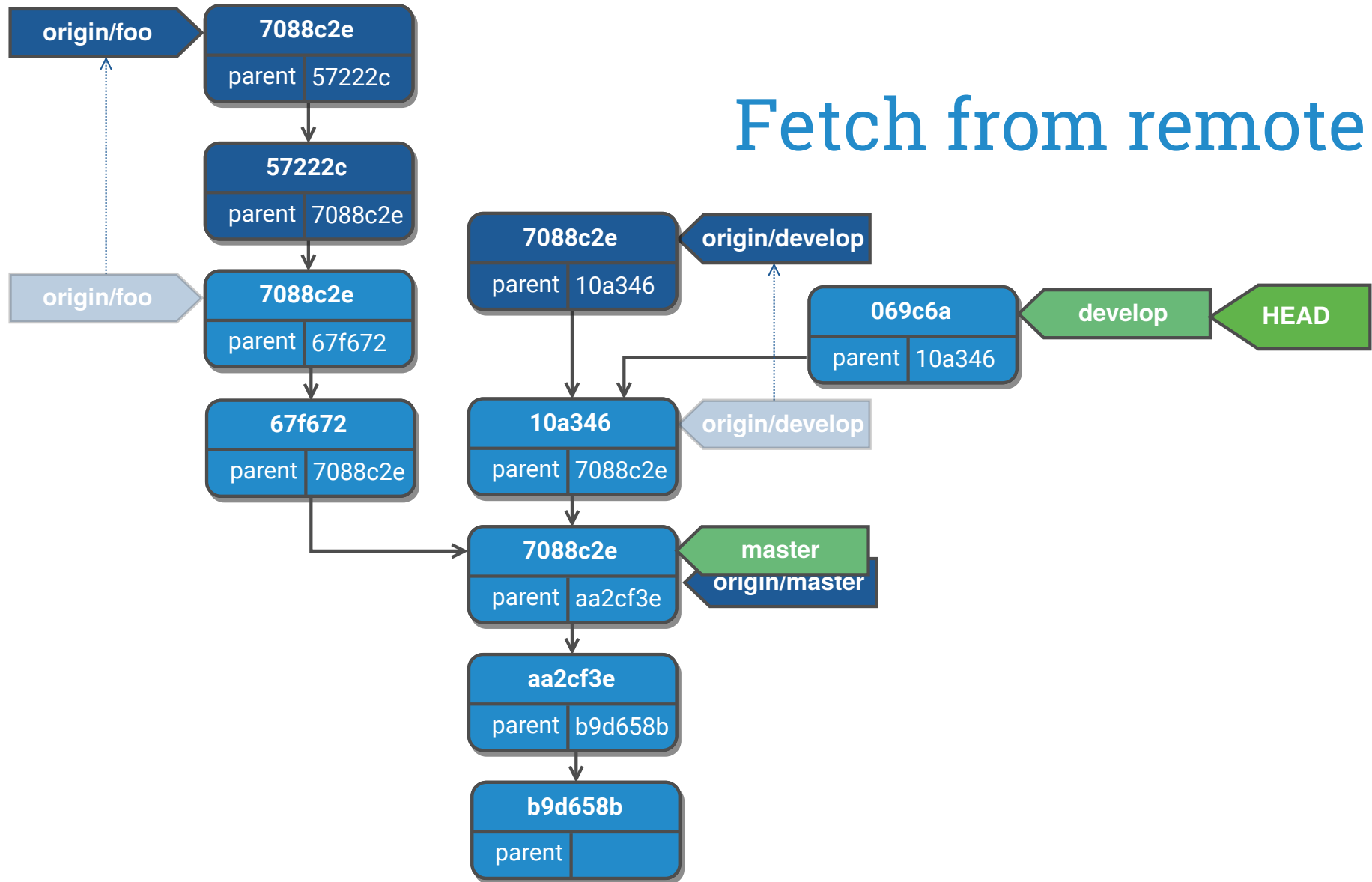
Adds the remote repository on the specified url under the given name.

If this is the first remote, **origin** is a good default for the name.

git remote show <name>

Show information about the remote repository (URL, tracked branches, stale branches, ...)

Fetch from remote



Fetch from remote

git fetch

fetches the changes from the remote repository, without applying the changes

git fetch --prune

prunes the branch markers for branches that have been removed remotely

```
$ git fetch
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3),
done.
Receiving objects ...
...
From git://example.com/foo.git
 10a346..7088c2e  develop -> origin/develop
 7288c2e..7088abd  foo    -> origin/foo
```


Pull the changes from remote

git pull

Short-hand command for **git fetch** followed by **git merge**

This will try to merge the remote tracking branch into the active HEAD.

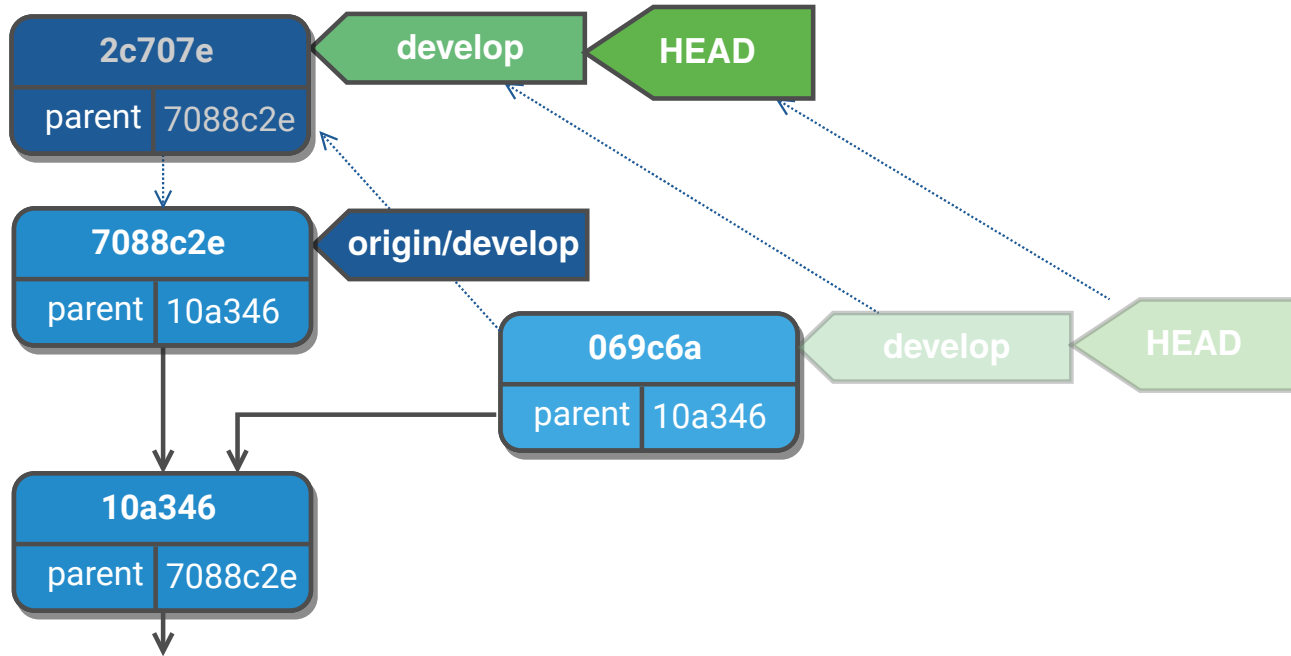
If both branches have commits, a merge-commit will be created

git pull -rebase

Short-hand command for **git fetch** followed by **git rebase**

This will rebase the active HEAD onto it's remote tracking branch

git rebase [origin/develop [develop]]

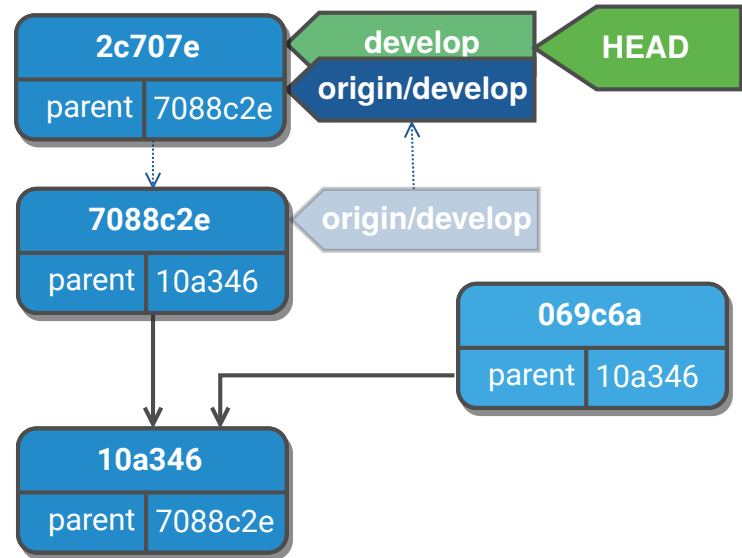


Push the changes

git push

pushes the commits from the local repository to the remote repository.

If you didn't push, your commits are just with you!



Find a remote branch

git branch

- r shows all remote branches
- a shows all local and remote branches

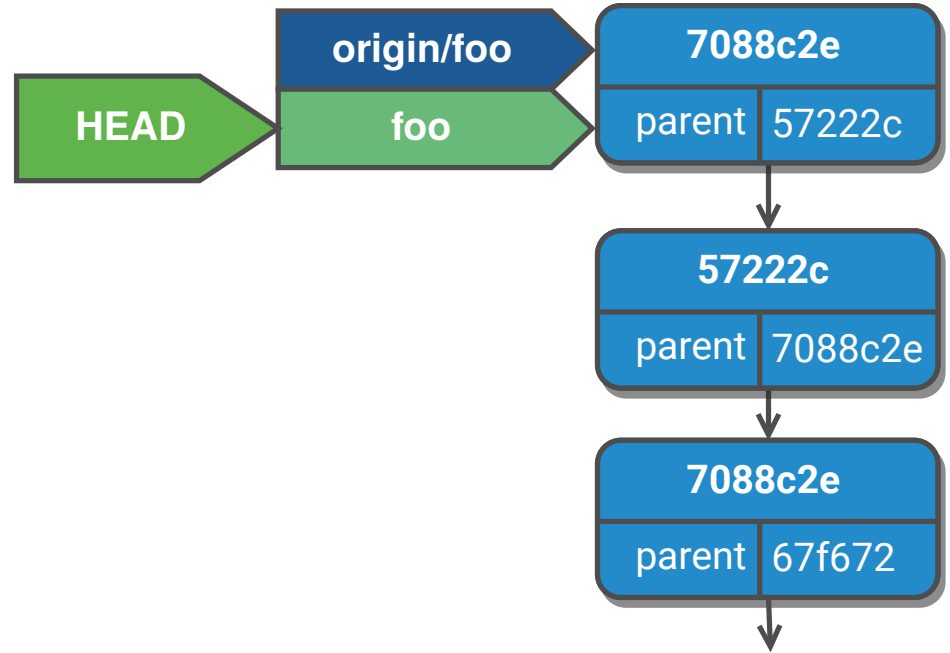
```
$ git branch -r
origin/HEAD -> origin/master
origin/develop
origin/foo

$ git branch -a
* develop
master
remotes/origin/HEAD -> origin/master
remotes/origin/develop
remotes/origin/foo
```

Checkout remote branch

git checkout <branch>

If a remote branch with the given name exists, a local tracking branch will be created and activated.



Create a remote branch

git push origin <branch>

creates a new remote branch with the specified name and the content of HEAD

git push origin <ref>:<branch>

creates a new remote branch with the specified name and the content of the specified reference

When using the **-u** flag, the new branch will also be configured as a tracking branch

Delete remote branch

git push --delete origin <branch>

git push origin :<branch>

Deletes the remote branch. Note that if a local branch exists, this local branch will not be deleted.



5

Workflow Strategies

Centralized Workflow

Everybody synchronizes with a single centralized branch.

- Compareable to a Centralized-VCS workflow

Detailed description:

<https://www.atlassian.com/git/tutorials/comparing-workflows/centralized-workflow>

Feature Branches

For every 'Story', a feature branch is created. This is where the work is done.

After the Story is completed, the feature branch can be merged into the master

- This can be merged directly

- This can also be merged using pull requests

Detailed description:

<https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>

Gitflow

A refinement of the Feature Branch workflow, using dedicated branches for production state and development code. Besides feature branches, it also knows release and hotfix branches.

Detailed description:

<http://nvie.com/posts/a-successful-git-branching-model/>

<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>



Resources

References & Cheatsheet

Official Git documentation : <https://git-scm.com/doc>

Git Reference : <http://gitref.org/>

GitHub Git Cheatsheet :

<https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf>

Visual Git Cheatsheet : <http://ndpsoftware.com/git-cheatsheet.html>

Learn Git Branching : <http://learngitbranching.js.org/>

Git SCM Wiki : <https://git.wiki.kernel.org/>

Video Resources

Official Git documentation : <https://git-scm.com/videos>

Knowledge is Power: Getting out of trouble by understanding Git

Steve Tarka : <https://www.youtube.com/watch?v=sevc6668cQ0>

Introductioni to Git with Scott Chacon of GitHub

<https://www.youtube.com/watch?v=ZDR433b0HJY>

Google Tech Talks :

Randal Schwartz on git (12.10.07) :<https://www.youtube.com/watch?v=8dhZ9BXQgc4>

Linus Torvalds on git (May 3, 2007)

<https://www.youtube.com/watch?v=4XpnKHJAok8>

Thank you!

