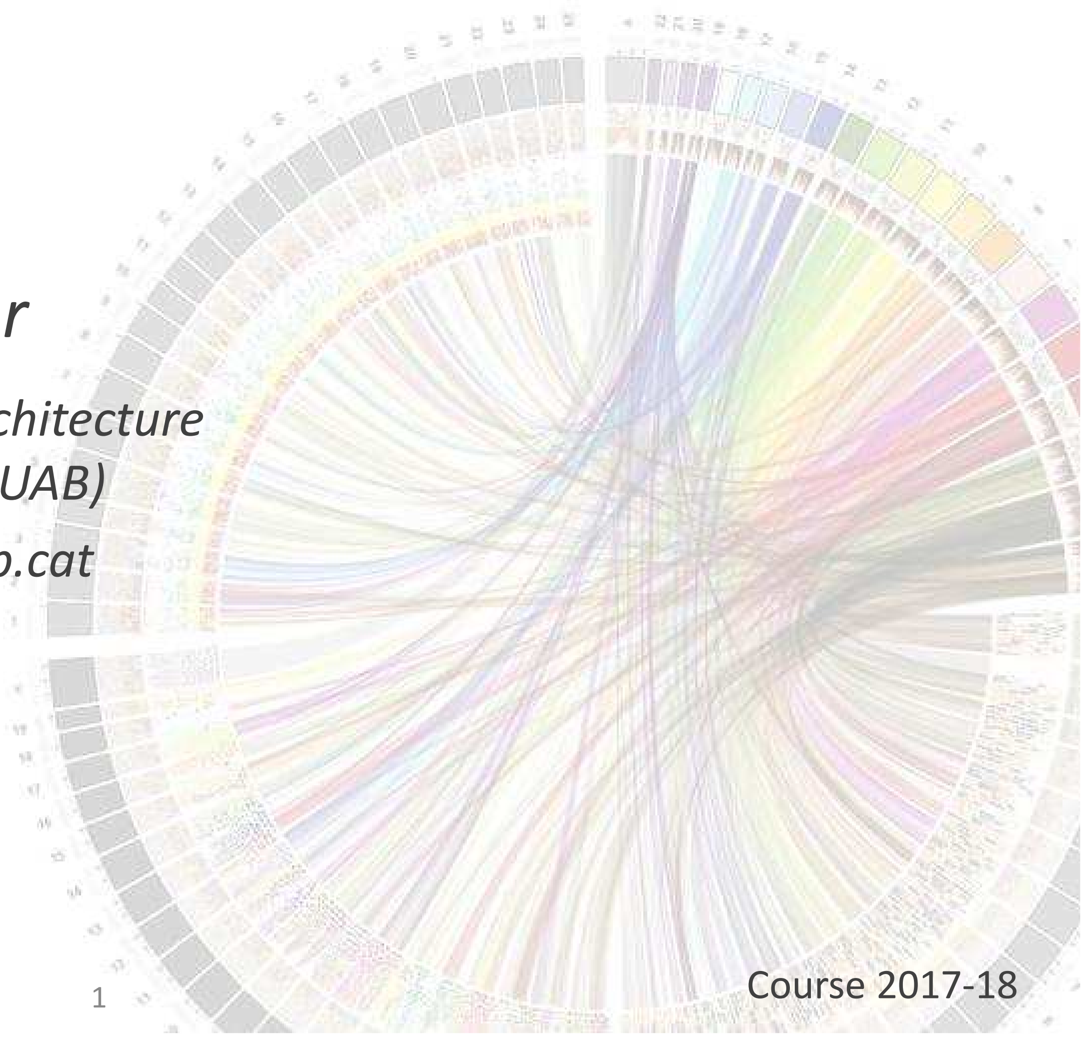
MSc in Bioinformatics

Module 2: Core Bioinformatics



GIT

Miquel A. Senar

Department of Computer Architecture and Operating Systems (UAB)

miquelangel.senar@uab.cat

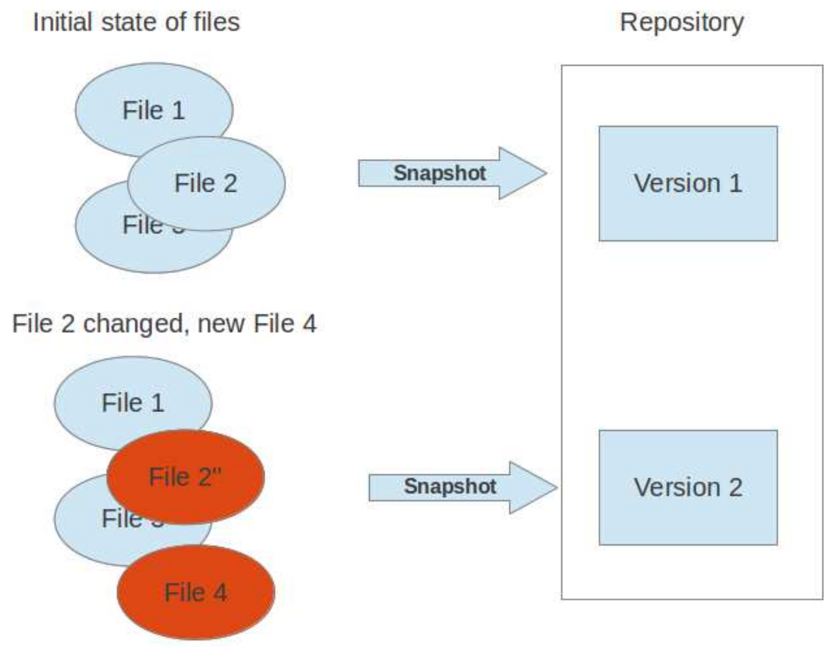
Course 2017-18

OUTLINE:

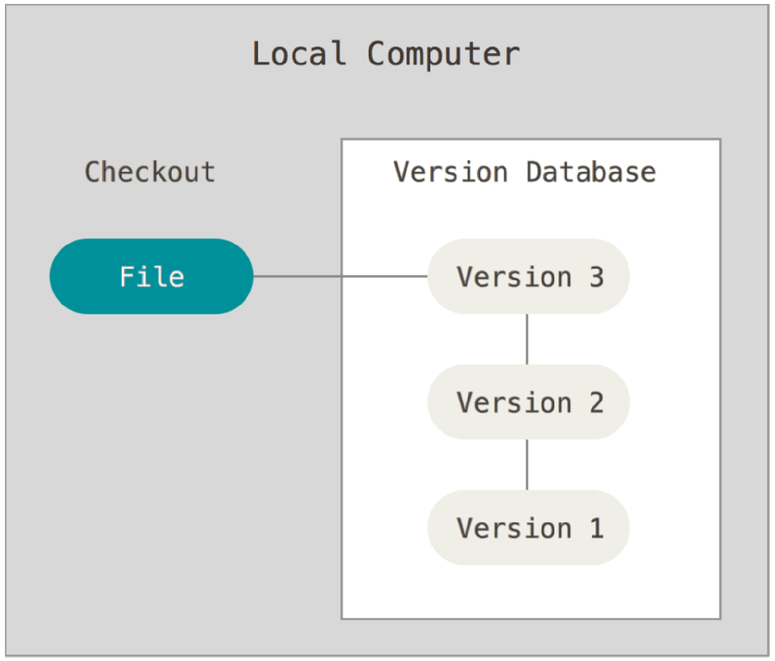
GIT

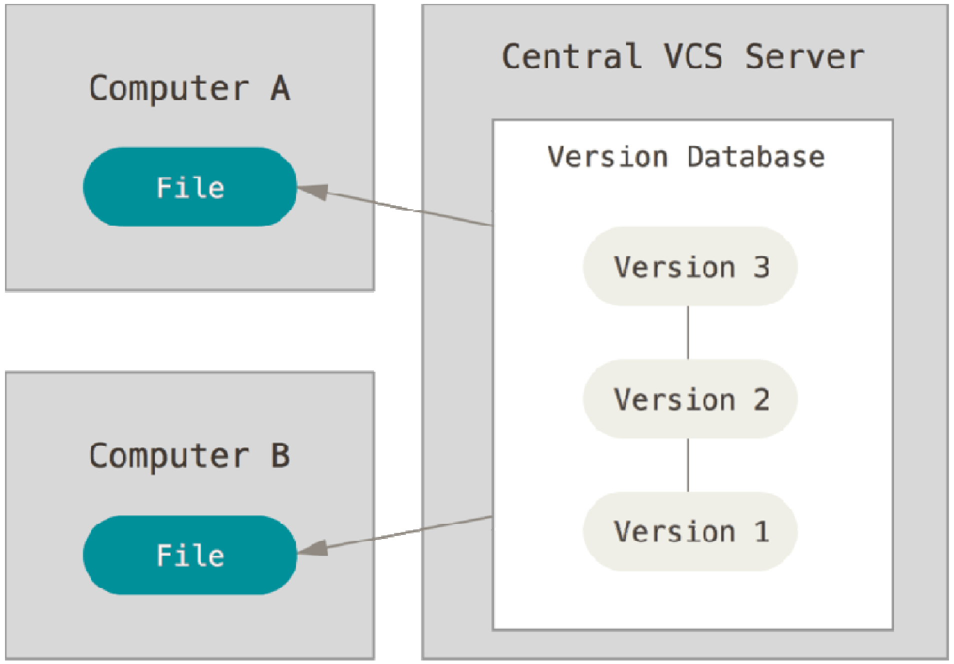
1. Version Control Systems

- A version control system allows you to track the history of a collection of files and includes the functionality to revert the collection of files to another version. Examples: CVS, Subversion, GIT.



1. Version Control Systems

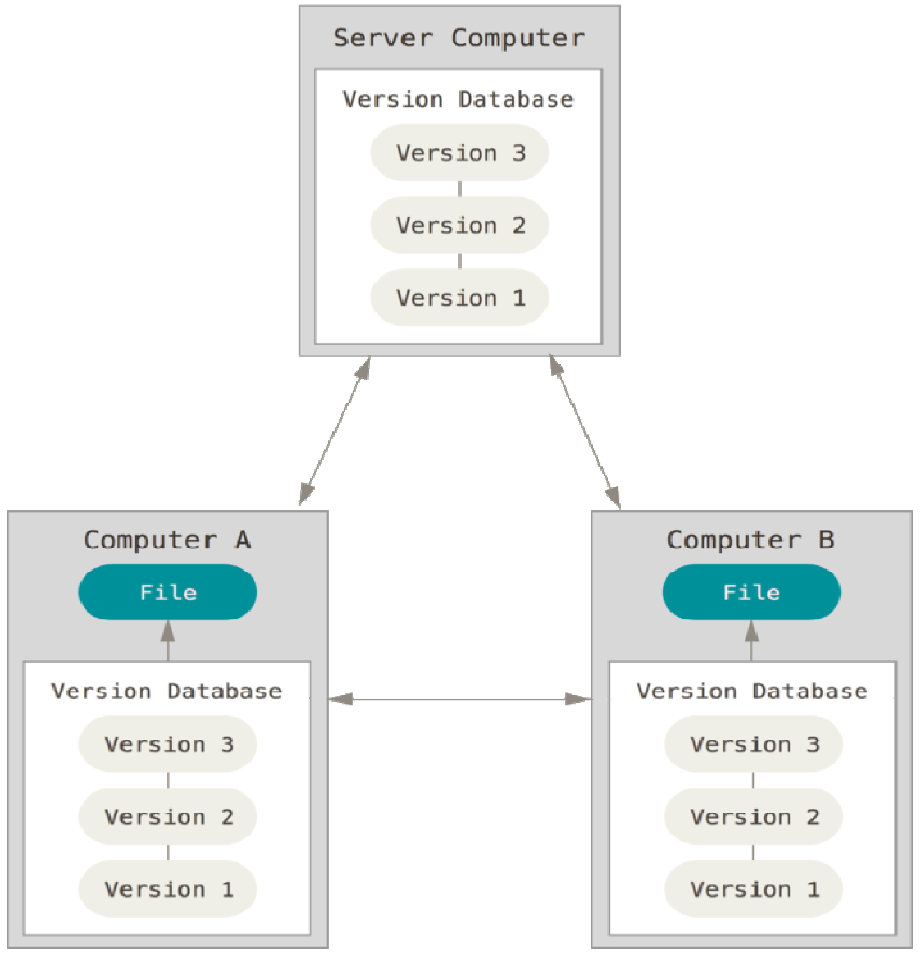




Local version control

Centralized version control

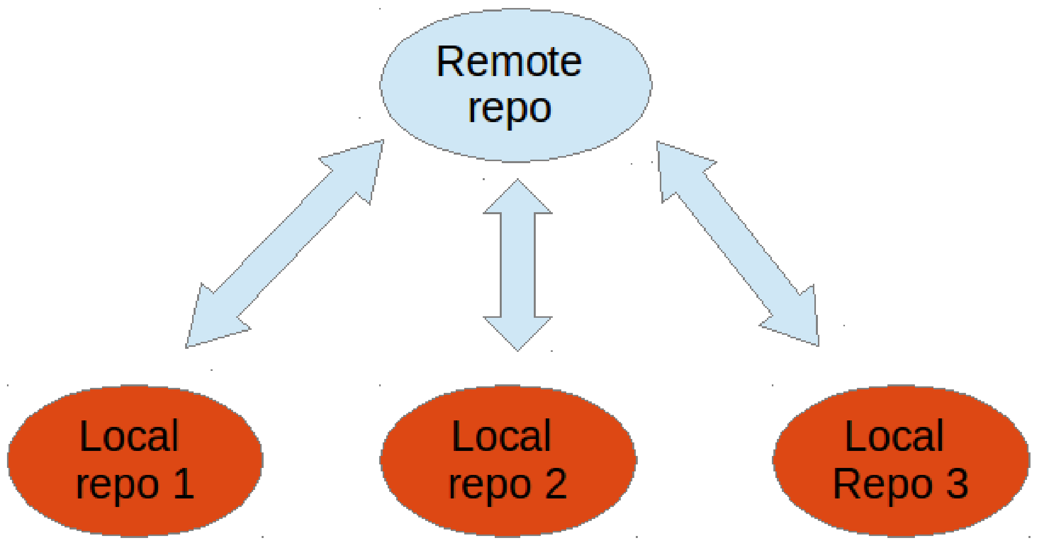
1. Version Control Systems



Distributed version control

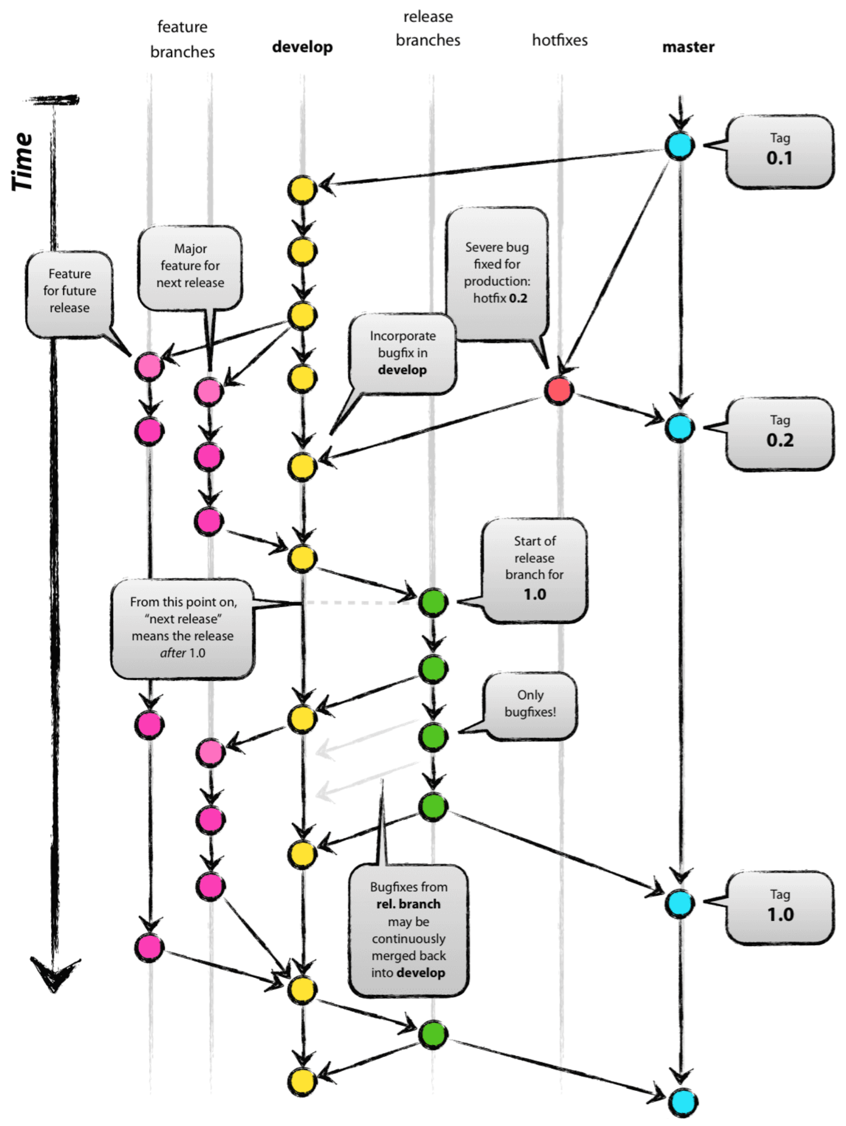
1. Distributed Version Control Systems

- A distributed version control system does not necessarily have a central server which stores the data.
- Typically there is a central server for keeping a repository but each cloned repository is a full copy of this repository.



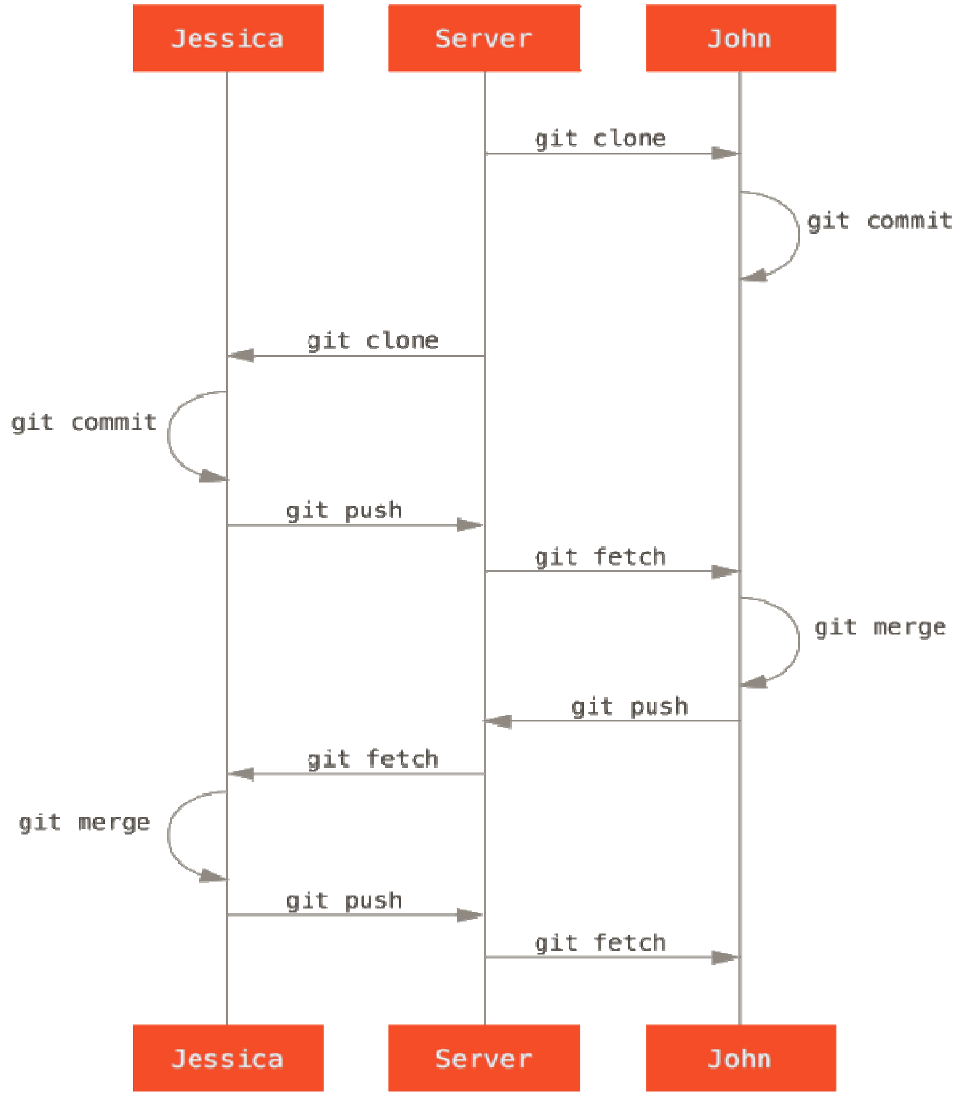
1. Distributed Version Control Systems

Development model based on a distributed VCS


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

1. Distributed Version Control Systems

General sequence of events for a simple multiple-developer Git workflow



Git for bioinformatics

- Git originally intended to work with large software projects.

- Complicated bioinformatic projects could benefit because:

  Git will track the evolution of small programs (C, C++), scripts (Python, Perl,…), R analyses, README files, latex papers,…

  Git will also help in collaborative work with many users.

- With Git, user generates a set of snapshots during project evolution that simplifies bug issolation by rolling back to past versions of the project.

2. What is Git and Basic Terminology

- *Git* is a distributed version control system.
- Git originates from the Linux kernel development and is used by many popular Open Source projects.
- The original tooling for Git was based on the command line. These days there is a huge variety of available Git tools (GUIs).
- A *repository* contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository.
- A *branch* is a named pointer to a commit.
- When you commit your changes into a repository this creates a new *commit object* in the Git repository.
- A *tag* points to a commit which uniquely identifies a version of the Git repository.

2. What is Git and Basic Terminology

Git repositories may be hosted on a local computer or on a central server

Hosting providers for Git

1. **GitHub**

   provides free hosting of publicly available Git repositories. If you want to have private repositories, which are only visible for selected people, you have to pay a monthly fee to GitHub (or GitHub Education pack that provides private repositories at a limited fee).

2. **Bitbucket**

   Bitbucket allows unlimited public and private repositories. The number of participants for a free private repository is currently limited to 5 collaborators, i.e., if you have more than 5 developers which need access to a private repository you have to pay.

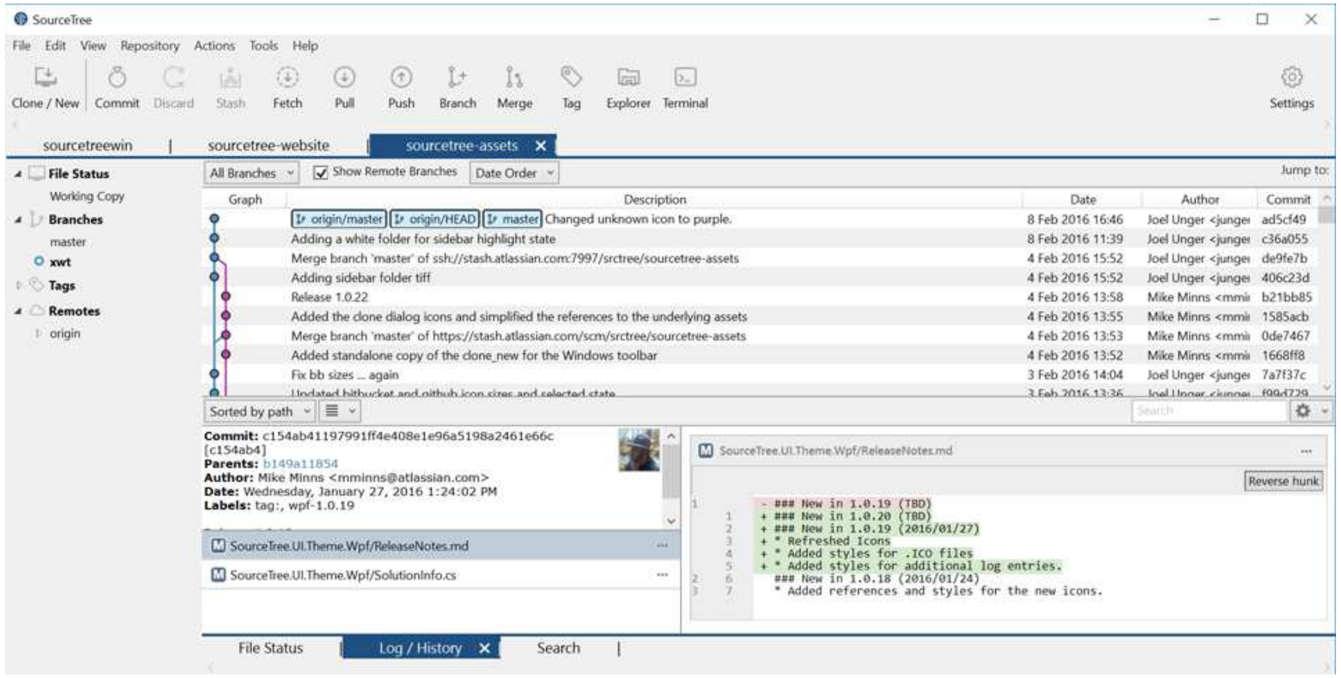
2. What is Git and Basic Terminology

Git can be used from the command line.

GUIs for Git

Linux: gitk, git gui, SmartGit, Eclipse EGit

Windows: Git for Windows, SmartGit , SourceTree, GitEye



Graph	Description	Date	Author	Commit
	origin/master origin/HEAD master Changed unknown icon to purple.	8 Feb 2016 16:46	Joel Unger <junger	ad5cf49
	Adding a white folder for sidebar highlight state	8 Feb 2016 11:39	Joel Unger <junger	c36a055
	Merge branch 'master' of ssh://stash.atlassian.com:7997/srctree/sourcetree-assets	4 Feb 2016 15:52	Joel Unger <junger	de9fe7b
	Adding sidebar folder tiff	4 Feb 2016 15:52	Joel Unger <junger	406c23d
	Release 1.0.22	4 Feb 2016 13:58	Mike Minns <mmi	b21bb85
	Added the clone dialog icons and simplified the references to the underlying assets	4 Feb 2016 13:55	Mike Minns <mmi	1585acb
	Merge branch 'master' of https://stash.atlassian.com/scm/srctree/sourcetree-assets	4 Feb 2016 13:53	Mike Minns <mmi	0de7467
	Added standalone copy of the clone_new for the Windows toolbar	4 Feb 2016 13:52	Mike Minns <mmi	1668ff8
	Fix bb sizes ... again	3 Feb 2016 14:04	Joel Unger <junger	7a7f37c
	Updated bitbucket and github icon sizes and selected state	3 Feb 2016 13:36	Joel Unger <junger	f99d729

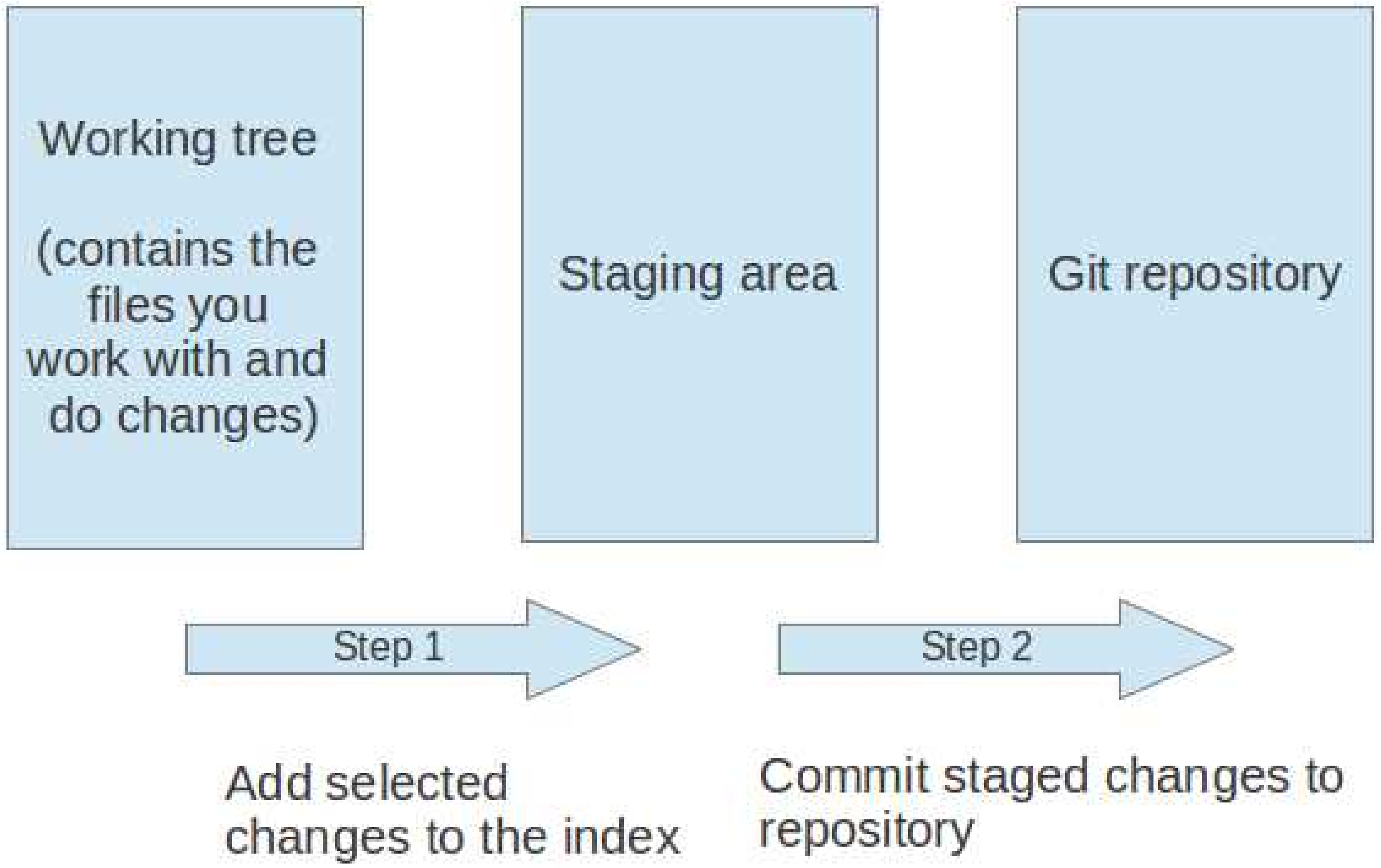
Commit: c154ab41197991ff4e408e1e96a5198a2461e66c
[c154ab4]
Parents: b149a11854
Author: Mike Minns <mminns@atlassian.com>
Date: Wednesday, January 27, 2016 1:24:02 PM
Labels: tag:, wpf-1.0.19

SourceTree.UI.Theme.Wpf/ReleaseNotes.md
SourceTree.UI.Theme.Wpf/SolutionInfo.cs

SourceTree.UI.Theme.Wpf/ReleaseNotes.md

```
- ### New in 1.0.19 (TBD)
+ ### New in 1.0.20 (TBD)
+ ### New in 1.0.19 (2016/01/27)
+ * Refreshed Icons
+ * Added styles for .ICO files
+ * Added styles for additional log entries.
  ### New in 1.0.18 (2016/01/24)
  * Added references and styles for the new icons.
```

3. Basic Operation


Working tree
(contains the files you work with and do changes)
Staging area
Git repository
Step 1
Step 2
Add selected changes to the index
Commit staged changes to repository

Starting with Git

4. Setting up a repository

- Installation as usual software (eg. apt-get install git in Ubuntu).
- Some configuration (.gitconfig file)

```
git config -- global user.name  “masenar”
git config -- global user.email “miquelangel.senar@uab.es”
```

.gitignore file: to ignore certain files and directories

note: in bioinformatic projects, large files (FASTA or FASTQ) should be ignored and not be included in the repository.

4. Starting up a repository

- Creation of a repository

  git init <directory> (developers local repository)

  git init -- bare <directory> (central repository)

- Cloning a repository

  git clone <repository> <directory> (clone the repository located at <repo> into the <directory> folder on the local machine)

Cloning a remote repository

- We will clone a repository from Github
- First, let’s take a look at github.com (at the repositories of Heng Li)

  go to https://github/lh3

  browse into seqtk
- Now, let’s clone seqtk into your local machine

  $git clone https://github.com/lh3/seqtk.git

  check the contents of seqtk directory (using *ls*). You could eventually compile the two aplications by doing *make*

Creating a local repository

- Create a directory *project_1* (*mkdir*)
- Enter into project_1 and create a directory *data*

```
$ cd project_1
$ mkdir data
```

- Create a file data_1.dat in directory *data*

```
$ cd data
$ echo “Sample 1: 1, 2, 3, 4, 5, 6, 7, 8, 9” > data_1.dat
$ cat data_1.dat
```

- Create a file README at directory project_1

```
$ echo “Project created by Paul” > README
$ cat README
```

4. Saving changes

- Adding changes in the working directory to the staging area.

  ```
  git add <file>
  git add <directory>
  ```

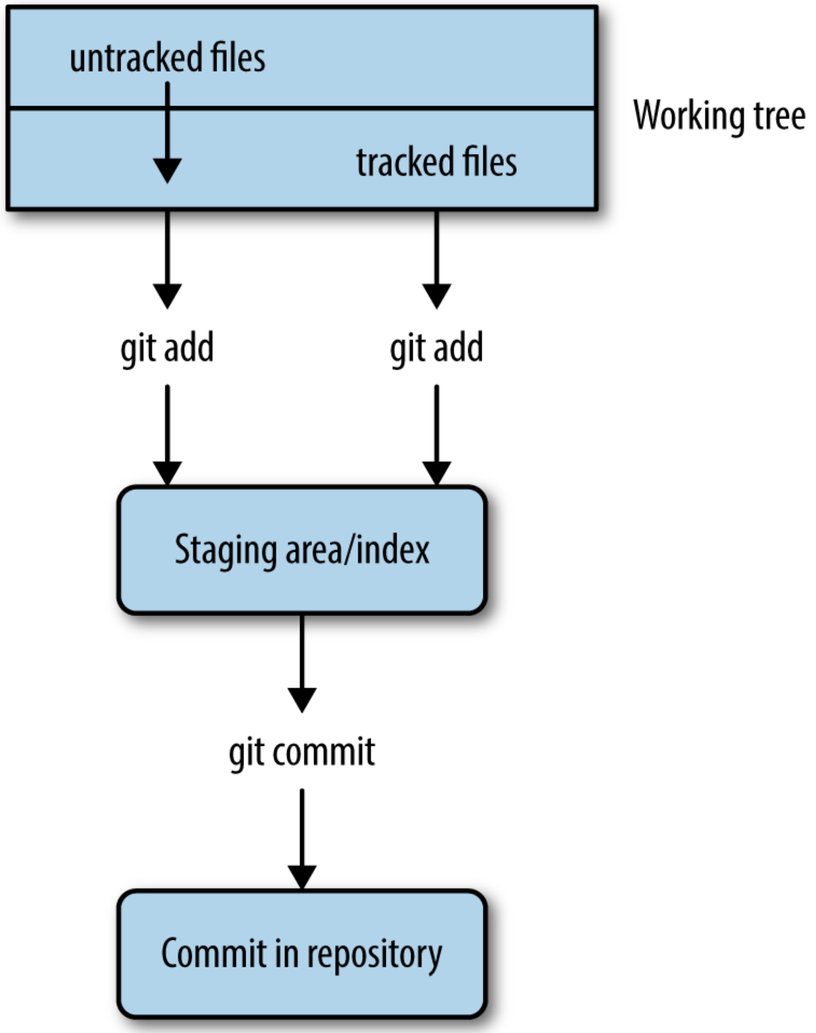
- Commiting the staged snapshot to the project history

  ```
  git commit
  git commit –a
  ```
  (commits all changes in the working directory)

- List of files staged, unstaged and untracked

  ```
  git status
  ```

Git’s working areas



Example 1

```
--- - switch to project_1
----  initialize local repository
$ git init
Initialized empty Git repository in /home/caos/youraccount/project_1/.git/

----- check the status of the repository
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README
#       data/data_1.dat
#
```

Example 1 (cont.)

```
--- - tracking files
$ git add README data/data_1.dat       ---- (equivalent to git add .)
---- check the  status of the repository
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
#       new file:   data/data_1.dat
#
```

Example 1 (cont.)

```
$ git commit –m “My first commit”
[master (root-commit) 976819f] First commit
 Committer: masenar <masenar@aolin21.uab.es>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

If the identity used for this commit is wrong, you can fix it with:

    git commit --amend --author='Your Name <you@example.com>'

 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 README
 create mode 100644 data/data_1.dat
```

5. Inspecting a repository

- Show the working tree status: list of files staged, unstaged and untracked

  git status

- Display the entire commit history

  git log

- Show changes between commits, commit and working tree, etc.

  git diff

Example 2

```
--- -  changing something at the repository
$ echo “I have added a second line at README” >> README
$ echo “User Manual” > Manual.txt
---- check the  status of the repository
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Manual.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

--- Update the repository including all changes (solution in next transparency but think about your answer first)

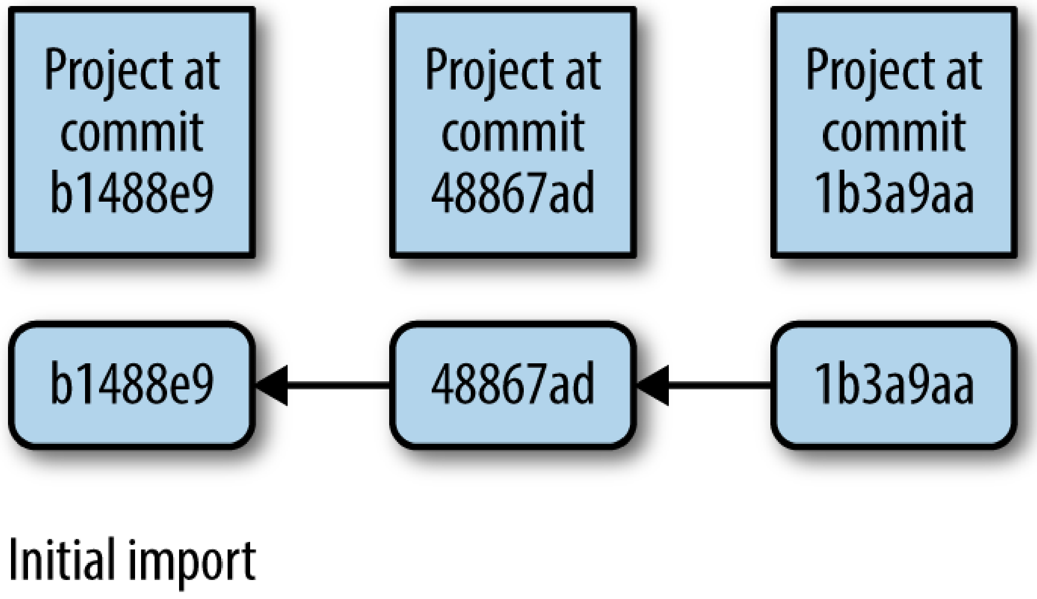
Example 2 (cont.)

```
$ git add .
$ git status    ----- not needed, but added here for the sake of clarity
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   Manual.txt
#       modified:   README
#
$ git commit –m "README modified and Manual.txt created"
[master d50fd4f] README modified and Manual.txt created
 Committer: masenar masenar@aolin21.uab.es
  ……………..
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 Manual.txt
```

Example 2 (cont.)

```
$ echo “Project started 04-11-2015”  >> README
$ git diff    -----  checking differences between files in working directory and what’s
              ----- been staged
diff --git a/README b/README
index 0dcb2eb..4a333f5 100644
--- a/README
+++ b/README
@@ -1,2 +1,3 @@
 Project created by Paul
 I have added a second line at README
+Project started 04-11-2015

$ git add README    ------  file added to the stage area
$ git diff          ------  shows nothing
$ git diff --staged ------ comparing what’s been staged to our last commit
```

Commit history


Project at commit b1488e9
Project at commit 48867ad
Project at commit 1b3a9aa
b1488e9
48867ad
1b3a9aa
Initial import

- Commits take discrete snapshots of our project at some point in time. Each one points to its parent, creating a chain that shows project evolution.

Example 2 (cont.)

```
$ git log
commit d50fd4f5eae994888ba764aa65c18baa678d87dd
Author: masenar <masenar@aolin21.uab.es>
Date:   Tue Nov 3 13:10:28 2015 +0100

    README modified and Manual.txt created

commit 976819f841967fd3a98d00273e30c6b792b043cc
Author: masenar <masenar@aolin21.uab.es>
Date:   Tue Nov 3 12:00:50 2015 +0100

    First commit
$ git log --graph --pretty=short
* commit d50fd4f5eae994888ba764aa65c18baa678d87dd
| Author: masenar <masenar@aolin21.uab.es>
|
|     README modified and Manual.txt created
|
* commit 976819f841967fd3a98d00273e30c6b792b043cc
  Author: masenar <masenar@aolin21.uab.es>

      First commit
```

Example 2 (cont.)

---- Comparing commits and files

```
$ git diff 976819f    ------ comparing what we have now and our first commit
diff --git a/Manual.txt b/Manual.txt
new file mode 100644
index 0000000..94f8662
--- /dev/null
+++ b/Manual.txt
@@ -0,0 +1 @@
+User Manual
diff --git a/README b/README
index 9a38e58..4a333f5 100644
--- a/README
+++ b/README
@@ -1 +1,3 @@
 Project created by Paul
+I have added a second line at README
+Project started 04-11-2015
```

Changing a repository

- Git wants to be in charge of tracked files. Using *mv* or *rm* commands will confuse it. Instead we have to use Git's versions of *mv* and *rm.*
- Delete a file from your working tree and record the deletion of the file in the staging area

```
git rm
```

- Move or rename a file or a directory from your working tree

```
git mv      (ex. git mv README README.md)
```

6. Viewing old commits and undoing changes

- Checking out an old version of working directory or a file

  git checkout <commit>

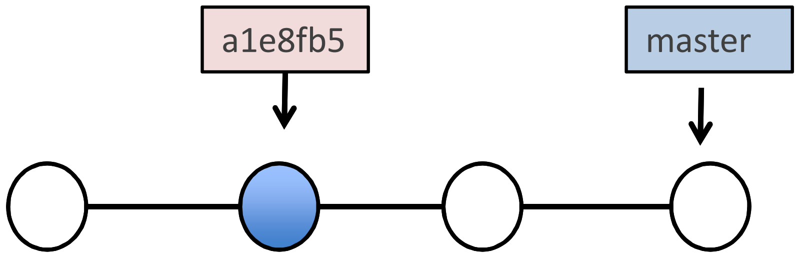
  git checkout <commit> <file>

```
$ more README
Project created by Paul
I have added a second line at README
$ echo "Added an accidental line" > README
$ more README
Added an accidental line
$ git checkout -- README    // “--” avoids potential confusion with a branch named README
$ more README
Project created by Paul
I have added a second line at README
```

6. Viewing old commits and undoing changes

- Returning to the master branch. A way to get back to the "current" state of the project.

```
git checkout master
```



6. Undoing a stage

- Messy changes don’t need to be included in a commit – we just don’t stage them.

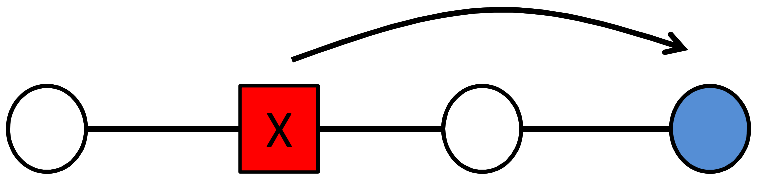
```
git  reset HEAD <file>
```

git reset command will reset HEAD to a specified state. When a <file> is specified, the file is removed from the staging area, but leaving the working directory unchanged.

6. Viewing old commits and undoing changes

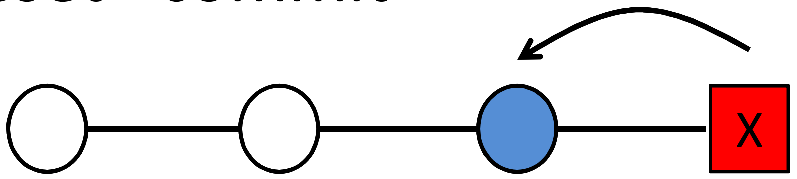
- Undoing a commited snapshot (safe way to undo changes). Generates a new commit that undoes all of the thancges introduced in <commit>, then apply it to the current branch.

```
git revert <commit>
```



- Removing commited snapshots (dangerous way to undo changes)

```
git reset <commit>
```



6. Undoing changes

- Removing untracked files from working directory

```
git  clean
```

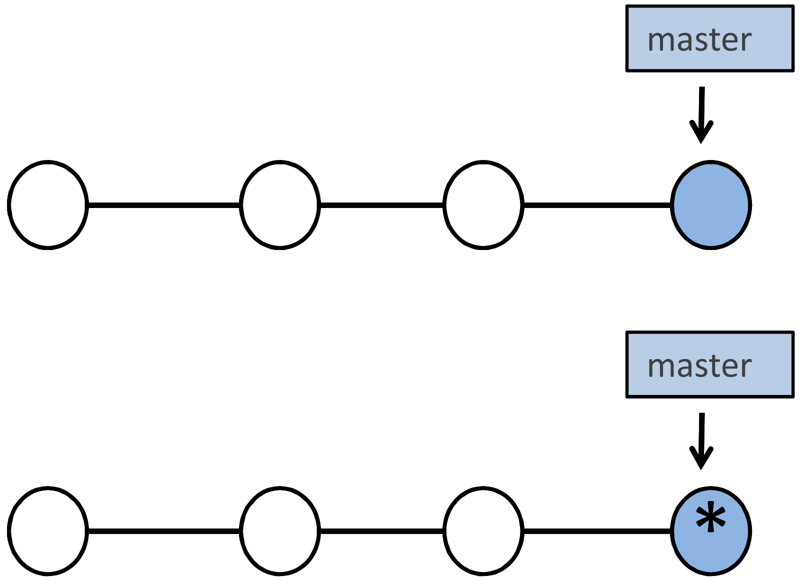
Not undoable (be careful). Try *git clean –n* first

git clean and git reset are very useful when you have made some embarrassing developments and you want to burn the evidence.

7. Rewriting history

- Fixing the most recent commit

```
git commit --amend
```



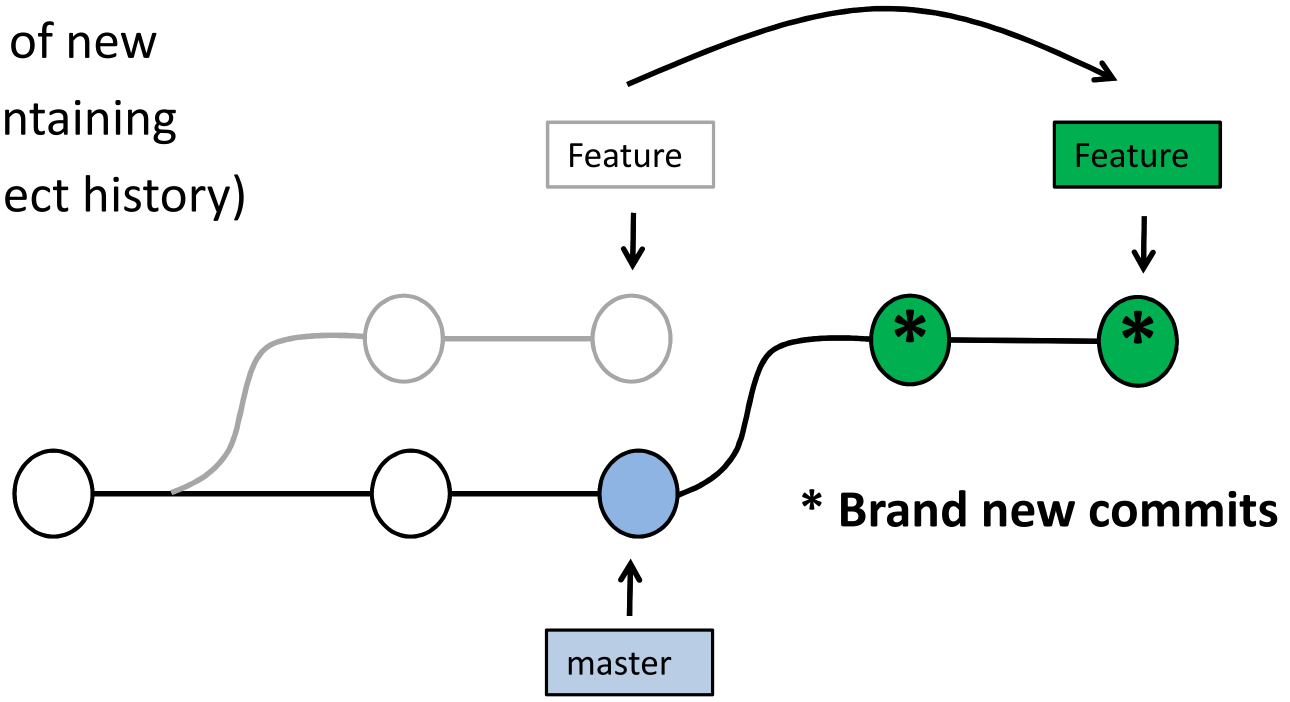
*** Brand new commits**

7. Rewriting history

- Moving a branch to a new base commit

  git rebase <base>

(integration of new features mantaining a linear project history)



Collaborating

8. Working with remote repositories

Git provides an easy way to access to remote repositories (central and co-workers) and work with other developers.

- Creating, viewing and deleting connections to other repositories.

git remote	lists the remote connections to other repositories
git remote add <name> <url>	creates a new connection
git remote rm <name>	removes the connection
git remote rename <old-name> <new-name>	renames a connection

8. Working with remote repositories

- When you clone a repository with **git clone**, it automatically creates a remote connection called origin pointing back to the cloned repository.
- Examples:

```
# clone online repository
git clone git://github.com/vogella/gitbook.git

# clone online repository
git clone ssh://git@github.com/vogella/gitbook.git

# the following will clone via HTTP
git clone http://github.com/vogella/gitbook.git
```

8. Working with remote repositories

- How to reference to a remote repository:
  - HTTP http://host/path/to/repo.git (read only usually)
  - HTTPS https://user@host/path/to/repo.git
  - SSH ssh://user@host/path/to/repo.git

    (read-write; requires valid SSH account)

- Importing commits from a remote repository to the local repo.

  git fetch <remote> fetch all branches

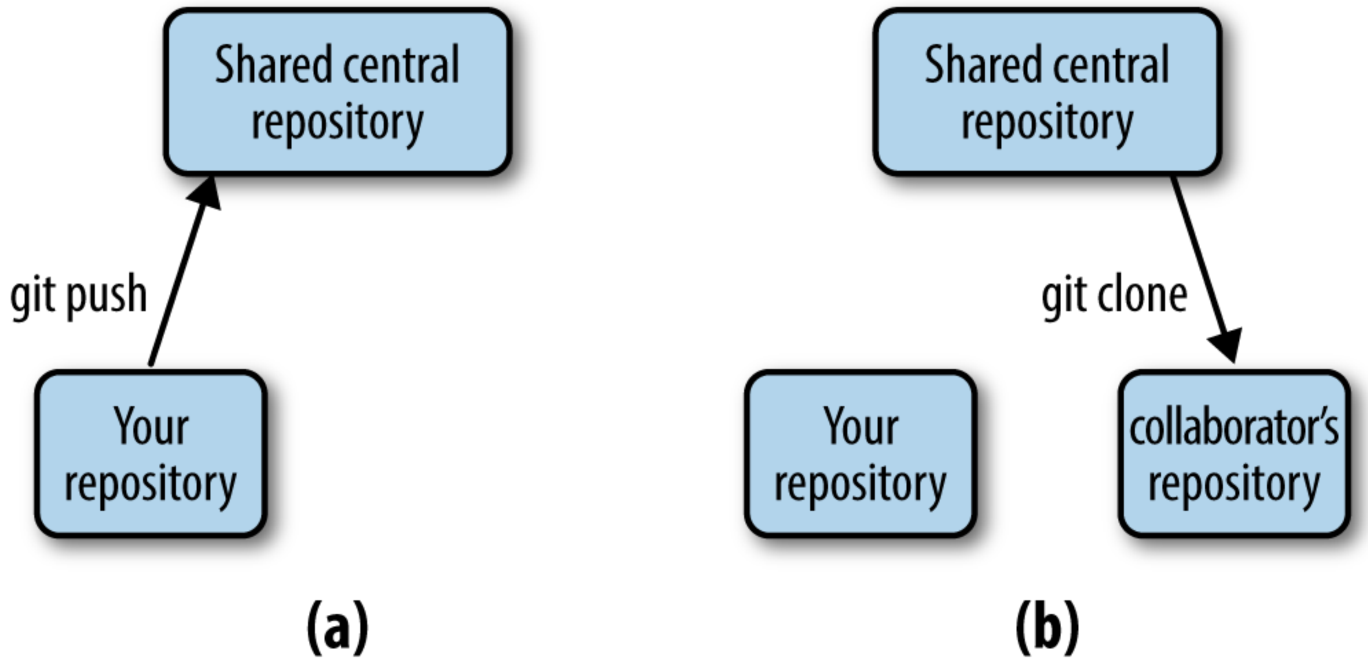
  git fetch <remote> <branch> fetch a specific branch

8. Working with remote repositories

- Fetching and merging a remote copy of a branch into the local copy
  - git pull <remote> uses git merge
  - git pull -- rebase <remote> uses git rebase

- Basic operations:
  - Create a shared central repository (accessible by all collaborators)
  - Push your project's initial commit
  - Collaborator clones initial work
  - Collaborator makes his/her changes to the project, commits the locally and then pushes to the CR
  - You pull collaborator's commit

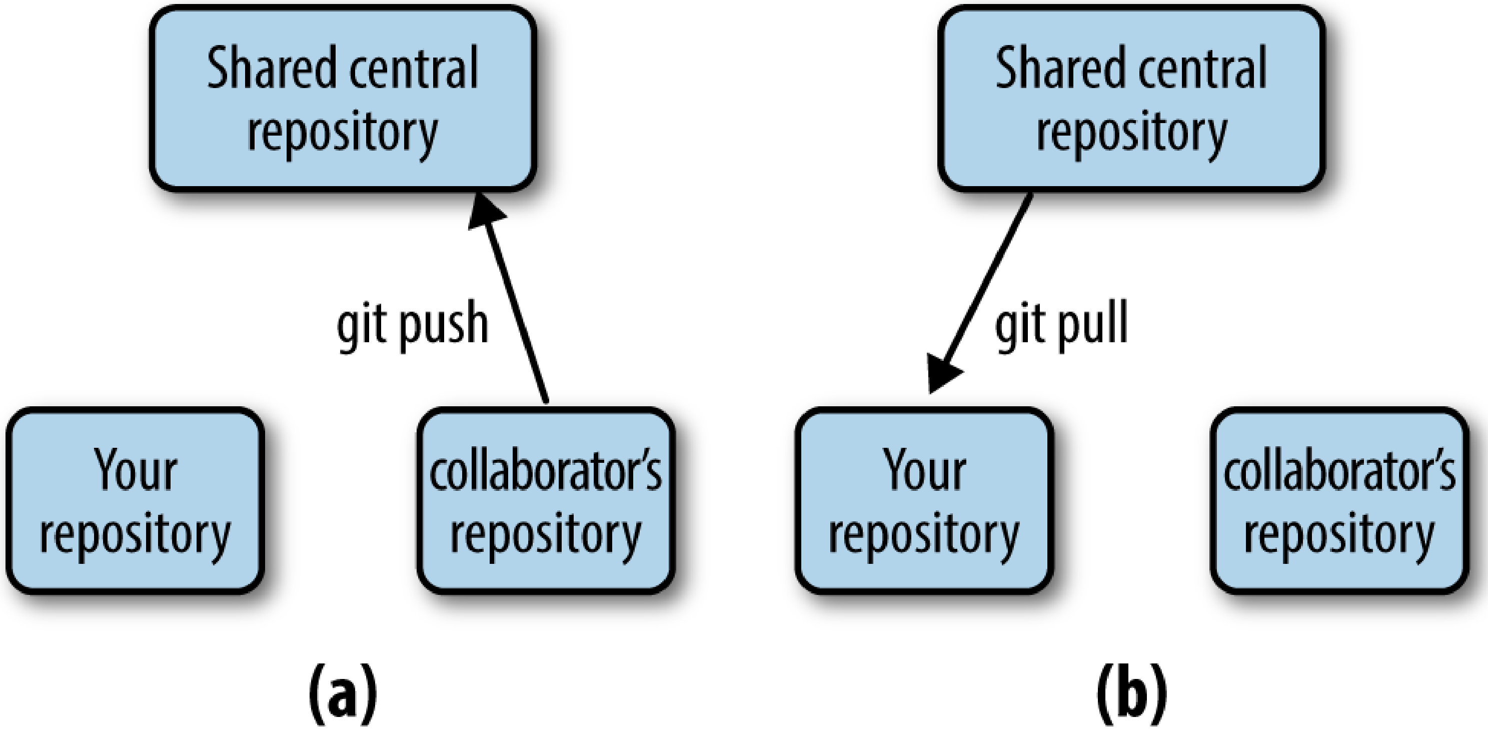
8. Working with remote repositories: basic operations


Shared central repository
git push
Your repository
(a)
Shared central repository
git clone
Your repository
collaborator's repository
(b)

(a) Creation of a new shared CR

(b) Collaborator retrieves project

8. Working with remote repositories: basic operations

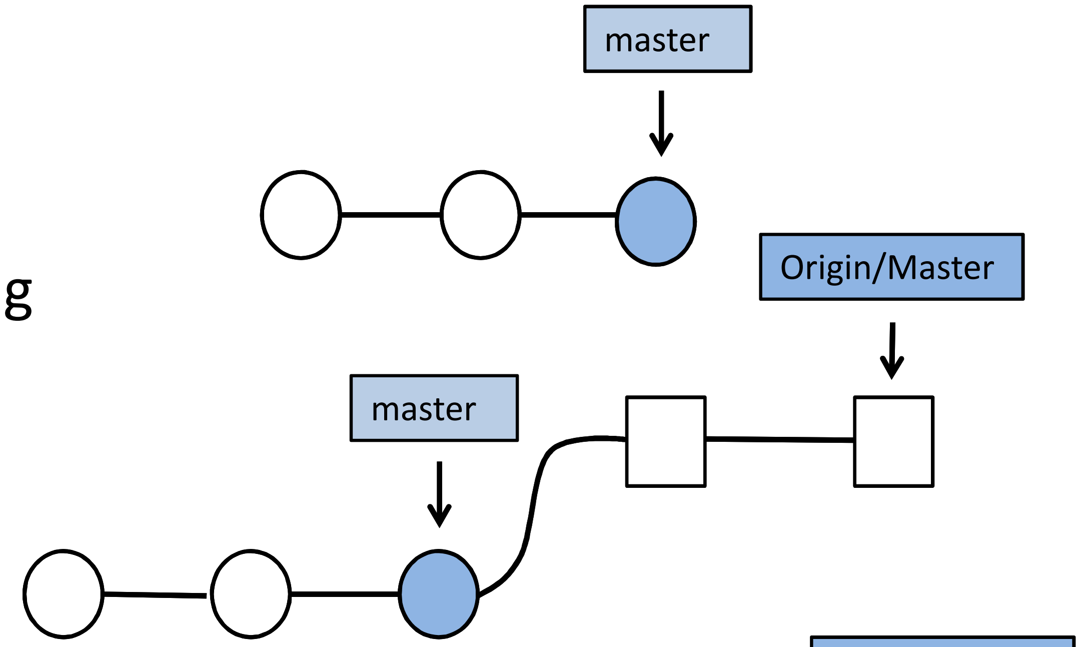

Shared central repository
git push
Your repository
collaborator's repository
(a)
Shared central repository
git pull
Your repository
collaborator's repository
(b)

(a) Collaborator pushes changes to CR

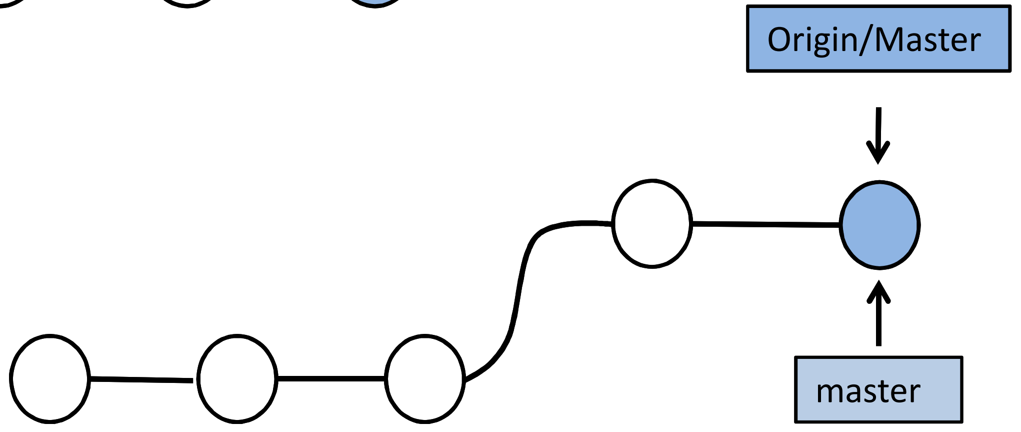
(b) Retrieving changes made by collaborator

8. Working with remote repositories

Before pulling



After pulling



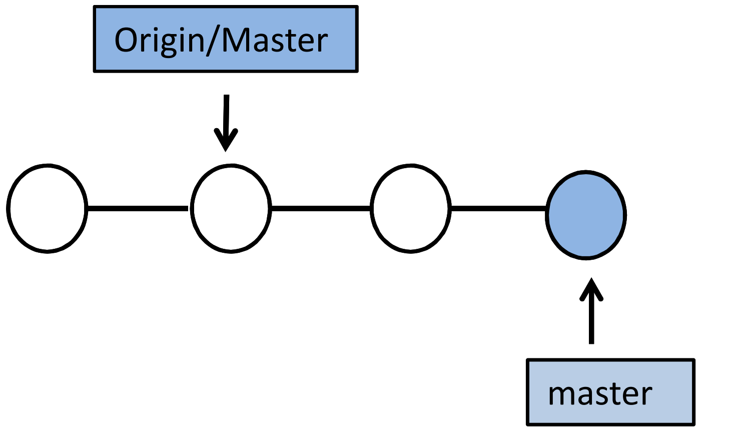
8. Working with remote repositories

- Transfering commits from local repository to a remote repository.

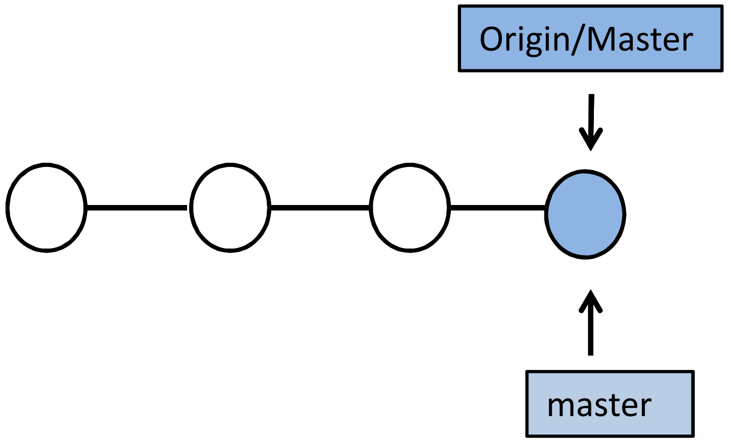
```
git push <remote>

git push <remote> branch     (creates a local branch at the
                              remote repo. Update must be a
                              fast-forward merge)
git push <remote> --force    (forces merge even it results in a
                              non-fast-forward merge)
                              Dangerous, unless you know
                              what you are doing
```

git push used tu publish local changes to central repository

8. Working with remote repositories



Before pushing



After pushing

8. Creating a shared central repository with GitHub

1. Go to github.com and sign up (for simplicity, pick the same username as the one you are using now)
2. Create a New repository with name project_1 (make sure it is marked as public)
3. Check the new repository from the main page.
4. Make sure all collaborators have a GitHub account.
5. Write access will be granted by adding *collaborators.*
6. GitHub uses SSH keys to authenticate users, preventing the need for entering a password each time (check the manual)

Example 3

```
$ git remote add origin https://youraccount@github.com/youraccount/project_1
```

--- our local repository project_1 will use the GitHub repository as a remote repository (it's name will be *origin*).

```
$ git remote –v
origin  https://youraccount@github.com/youraccount/project_1 (fetch)
origin  https://youraccount@github.com/youraccount/project_1 (push)

$unset SSH_ASKPASS        ----  if needed to prevent the bash shell to launch a dialogue box
$ git push origin master   ----  pushing our local repository to GitHub
Password:
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 407 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://youraccount@github.com/youraccount/project_1
  976819f..d50fd4f  master -> master
```

Example 3 (cont.)

```
--- cloning to a fake collaborator machine from project_1
$ git clone git://github.com/youraccount/project_1   ../collaborator_project_1
Initialized empty Git repository in /home/caos/youraccount/collaborator_project_1/.git/
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.


--- now, you can check the contents of directory  collaborator_project_1
--- in collaborator_project_1 (collaborator’s repository).
$ echo “Collaborator added new comments in README” >> README
$ git commit -a -m "added new comments (Collaborator)“              ----- add and commit in one step
[master c7c0d3e] added new comments (Collaborator)
 Committer: masenar masenar@aolin21.uab.es     -----  (should be collaborator’s username in a real scenario)
…….
 1 files changed, 1 insertions(+), 0 deletions(-)
```

Example 3 (cont.)

```
$ git push  https://masenar@github.com/masenar/project_1.git master
    -----  ( or git push origin master, if SSH keys are used)
Password:
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 401 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://masenar@github.com/masenar/project_1.git
   d50fd4f..c7c0d3e  master -> master

---- At our original repository, we see that README has been modified
---- (need to cd../project_1, for instance, to go bacj to the right directory)

$ git log --oneline origin/master      -----  we check commits at central repository (one line format)
c7c0d3e added new comments (Collaborator)
d50fd4f README modified and Manual.txt created
976819f First commit
```

Example 3 (cont.)

```
---- suppose we have already made same changes to our local repository
$ echo “Project started 04-11-2015” >> README          ---- third line of README differs from CR
$ git commit -a -m "Added project starting date“          ----  we commit our change
[master 124be36] Added project starting date
 Committer: masenar <masenar@aolin21.uab.es>

-----   we try to push it to CR
$ git push origin master
Password:
To https://masenar@github.com/masenar/project_1
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'https://masenar@github.com/masenar/project_1'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes before pushing again.  See the 'Note about
fast-forwards' section of 'git push --help' for details.
```

--- push was rejected due to a conflict with existing files at CR. As suggested, we have to merge before pushing

Example 3 (cont.)

```
----  We pull from the CR to get modified files
$ git pull origin master
Password:
From https://github.com/masenar/project_1
 * branch            master     -> FETCH_HEAD
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
----  with git status we check which files were modified and cause the conflict
$ git status
# On branch master
# Your branch and 'origin/master' have diverged,
# and have 1 and 1 different commit(s) each, respectively.
#
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      README                        -----  this is the file that produces the conflict
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Example 3 (cont.)

```
$ cat  README                         ----- We take a look at the file that produces the conflict
Project created by Paul
I have added a second line at README
<<<<<<< HEAD                                   ------  Start of our version
Project started 04-11-2015                     ------   Our line
=======                                        ------  Start of collaborator's changes
Collaborator added new comments in README      -----   Collaborator's line
>>>>>>> c7c0d3edc5a6cd0a23e38327621f37423e570067

----  We have to fix the problem by manually editing the file. For instance, README may look like
$ cat README
Project created by Paul
I have added a second line at README
Project started 04-11-2015                      ---- our line goes first

Collaborator added new comments in README       ----  collaborator's line goes here

$ git add README                                ----  staging the file to declare that conflict is resolved
```

Example 3 (cont.)

```
$ git status                    -----  checking that conflicts have been resolved and are ready to commit
# On branch master
# Your branch and 'origin/master' have diverged,
# and have 1 and 1 different commit(s) each, respectively.
#
# Changes to be committed:
#
#       modified:   README
#

$ git commit -m "resolved merge conflict in README"
[master 7726932] resolved merge conflict in README
 Committer: masenar <masenar@aolin21.uab.es>

$ git push origin master     -----  pushing our merge to central repository (can be checked at GitHub)
```

Example 3 (cont.)

$ git log --graph --- taking a look at the commit history with a graphical format

```
*   commit 772693205fa1ad6c10ae2cce3ca513d137db1e82
|\  Merge: 124be36 c7c0d3e                          ---- this is a merge commit; special type because it has two
| | Author: masenar masenar@aolin21.uab.es          ----  parents (our version and collaborator's version, which
| | Date:   Tue Nov 3 23:49:29 2015 +0100           ----  were based on a common ancestor)
| |
| |     resolved merge conflict in README
| |
| * commit c7c0d3edc5a6cd0a23e38327621f37423e570067
| | Author: masenar <masenar@aolin21.uab.es>
| | Date:   Tue Nov 3 22:39:08 2015 +0100
| |
| |     added new comments (Collaborator)
| |
* | commit 124be3694b4e392d2123fc3d41a5a26ffe79814b
|/  Author: masenar <masenar@aolin21.uab.es>
|   Date:   Tue Nov 3 23:23:18 2015 +0100
|
|       Added project starting date
|
* commit d50fd4f5eae994888ba764aa65c18baa678d87dd
| Author: masenar <masenar@aolin21.uab.es>
| Date:   Tue Nov 3 13:10:28 2015 +0100
|
|     README modified and Manual.txt created
|
* commit 976819f841967fd3a98d00273e30c6b792b043cc
  Author: masenar <masenar@aolin21.uab.es>
  Date:   Tue Nov 3 12:00:50 2015 +0100

     First commit
```

9. Using branches and merging

A branch represents an independent line of development (a brand new working directory, staging area and project history).

For example, trying a new pipeline in the middle of a variant calling project (if it does not work, it will not affect main project)

Developing new (software) features or bug fixes without affecting the working production version.

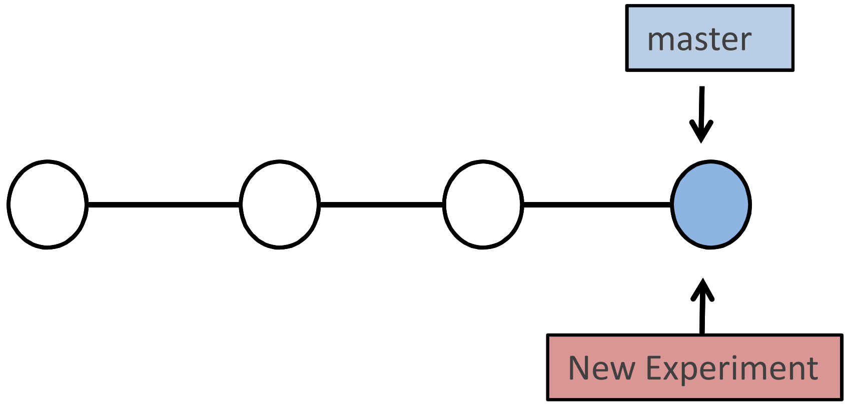
- Creation, listing, renaming, deleting branches

git branch	list
git branch <branch>	create
git branch –d <branch>	delete
git branch –D <branch>	force delete
git branch –m <branch>	rename current branch

9. Using branches and merging

- Navigating between branches

```
git checkout <existing-branch>
```

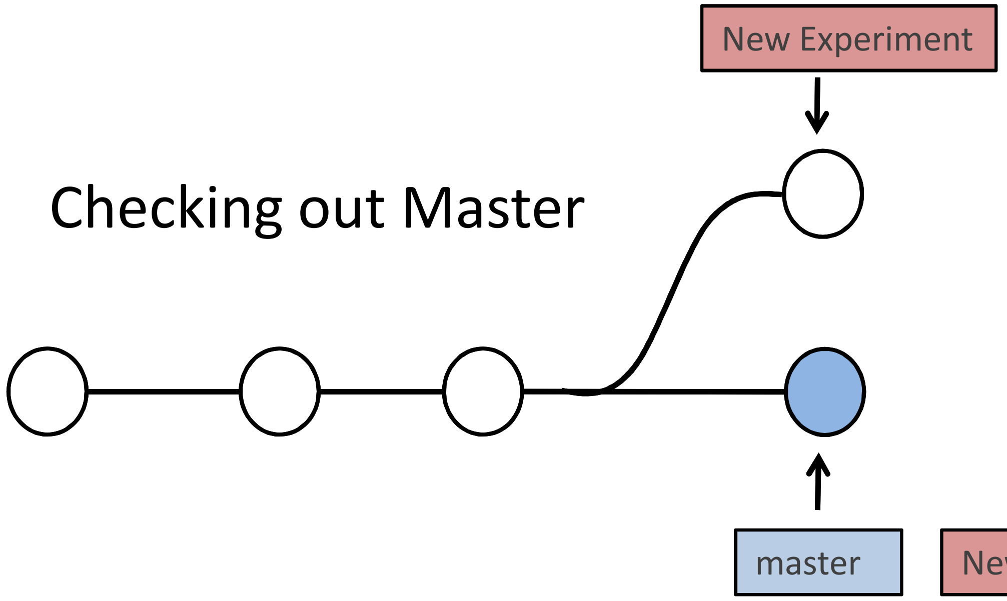


```
git branch New-Experiment
git checkout New-Experiment
```

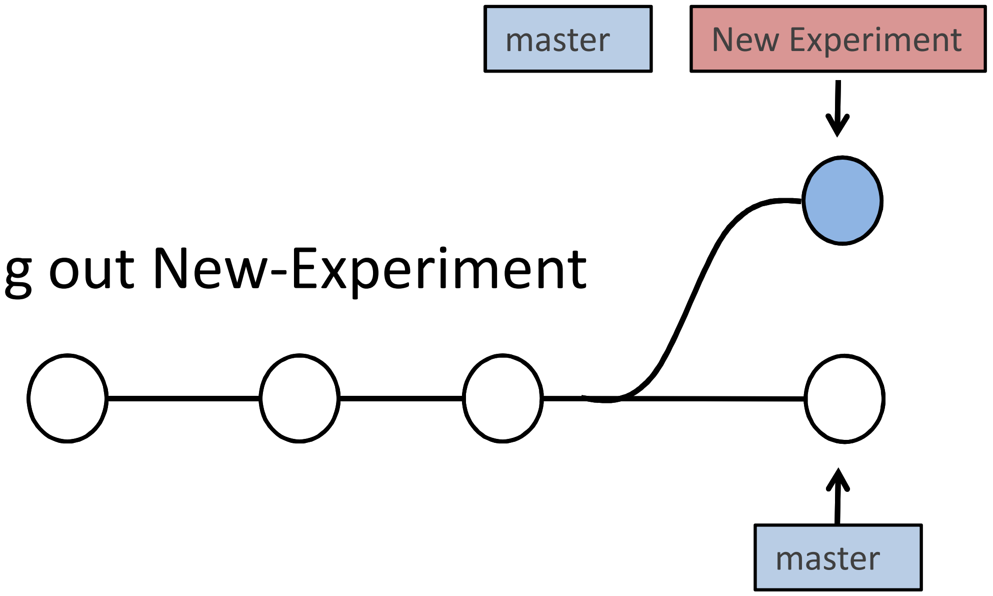
(new commits can be added now on this branch)

9. Using branches and merging

Checking out Master



Checking out New-Experiment



Example 4

```
$ git branch readme-changes
$ git branch
* master
  readme-changes
$ git checkout readme-changes
Switched to branch 'readme-changes'
$ git branch
  master
•     readme-changes
//  we edit README on the new branch
          Project created by Paul
          I have added a second line at README
          Project started 04-11-2015

          Collaborator added new comments in README

          # Added new samples
            data/seqs/A_R1.fastq
            data/seqs/A_R2.fastq
            data/seqs/A_R3.fastq
            data/seqs/A_r4.fastq
```

Example 4

```
$ git commit -a -m "New README with new samples"
[readme-changes badf5fd] New README with new samples
 1 files changed, 6 insertions(+), 0 deletions(-)
$ git log --abbrev-commit --pretty=oneline                    // looking at commits in readme-changes branch
badf5fd New README with new samples
7726932 resolved merge conflict in README
124be36 Added project starting date
c7c0d3e added new comments (Collaborator)
d50fd4f README modified and Manual.txt created
976819f First commit

$ git checkout master                                 //switching back to master branch
Switched to branch 'master‘
$ git log --abbrev-commit --pretty=oneline            // commits in master branch  don’t include the
7726932 resolved merge conflict in README             // commit done at readme-changes branch
124be36 Added project starting date
c7c0d3e added new comments (Collaborator)
d50fd4f README modified and Manual.txt created
976819f First commit
```

9. Using branches and merging

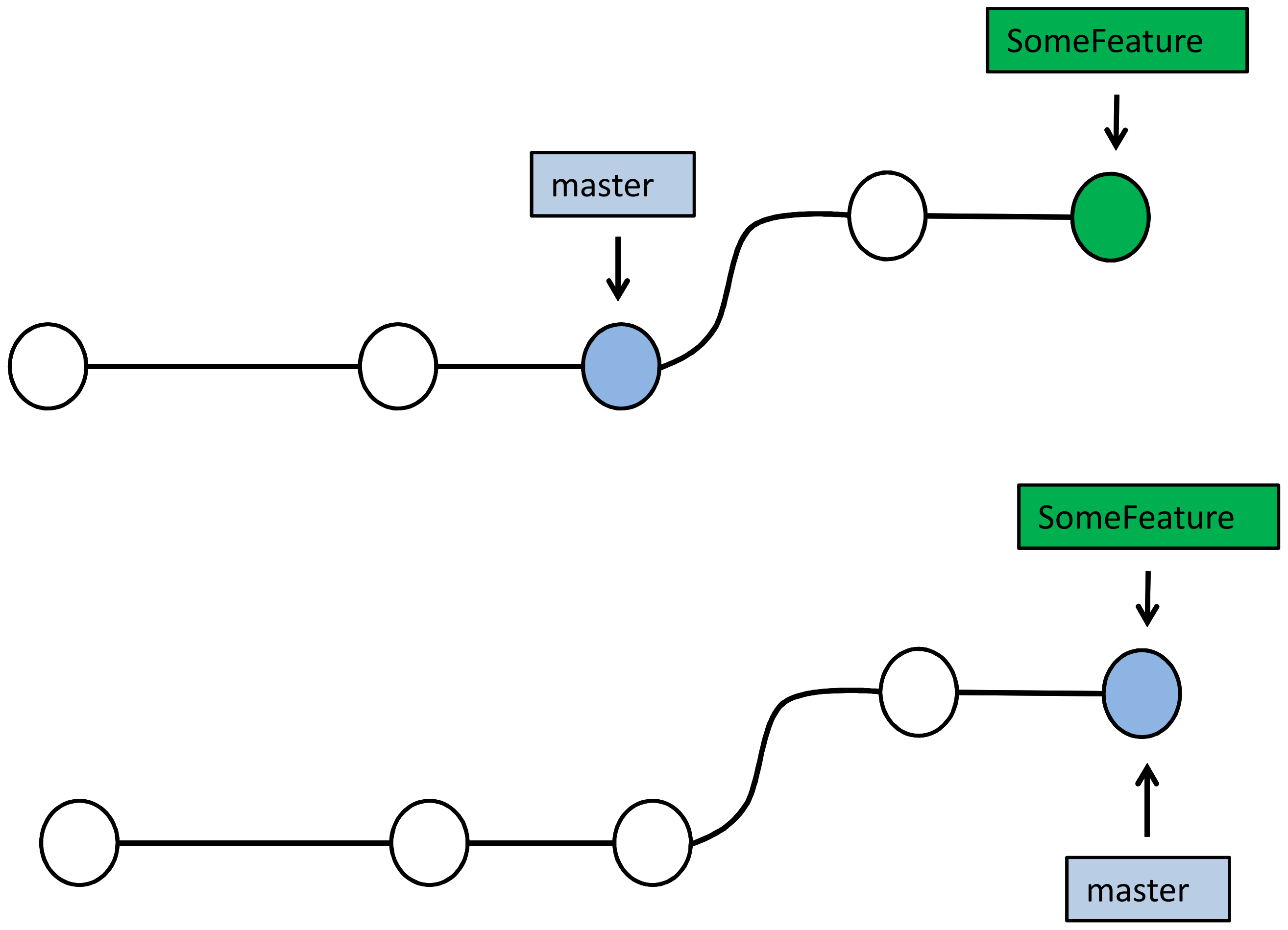
- Putting a forked history back together again.

```
git merge <branch>
```

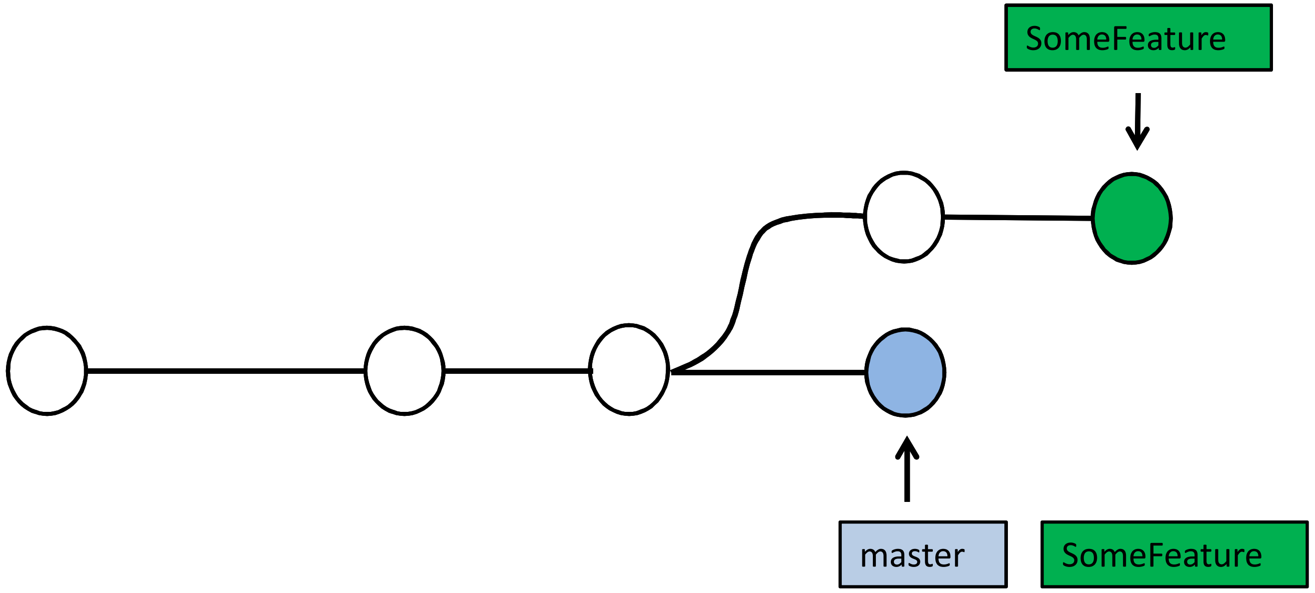
A **fast-forward merge** can occur when there is a linear path from the current branch tip to the target branch.

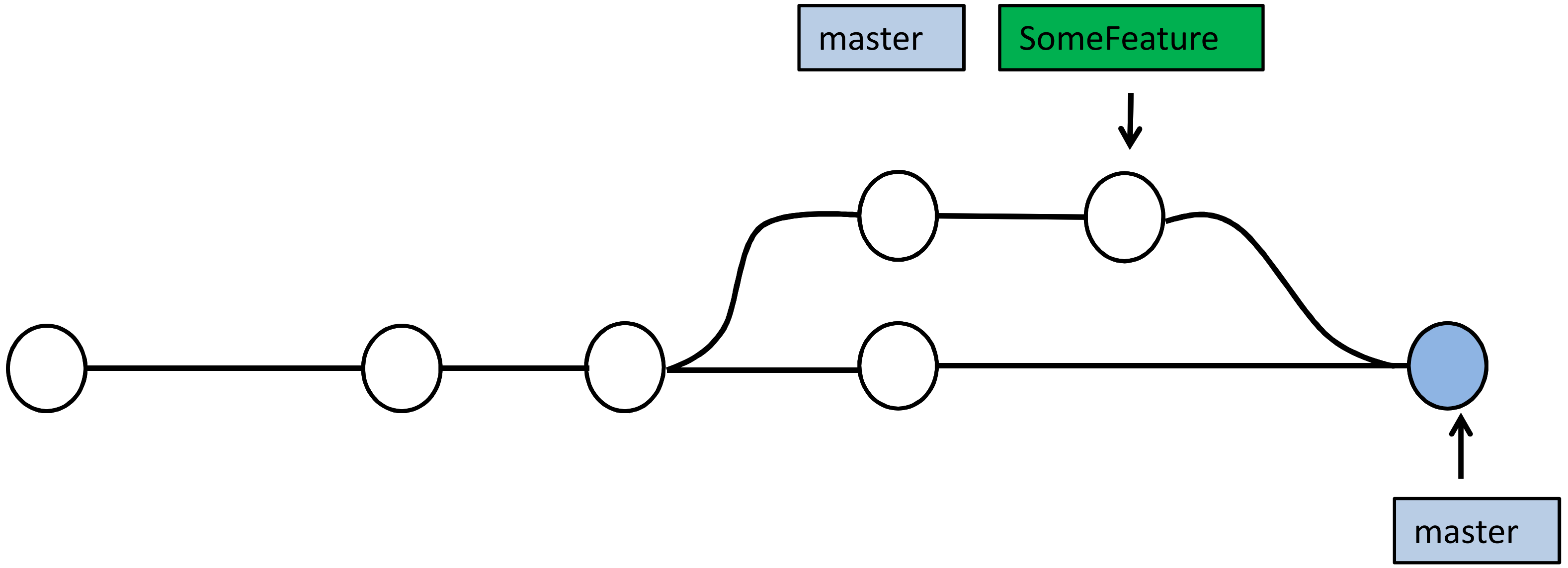
A **3-way merge** occurs when there is not a linear path and a dedicated commit is used to tie together the two histories.

Fast forward merge



3-way merge





Example 4 (3-way merge)

```
$ git branch
* master
  readme-changes
$ echo "Samples: 10, 20, 30, 40, 50" > samples.dat
$ git add samples.dat
$ git commit -m "added samples file"
[master 49a695c] added samples file
1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 samples.dat
```

```
#Merge in the new-feature branch
$ git merge readme-changes
Merge made by recursive.
 README |    6 ++++++
 1 files changed, 6 insertions(+), 0 deletions(-)
```

```
# we look at the commit history

$ git log --abbrev-commit --pretty=oneline --graph --
      branches
*   ea6ec69 Merge branch 'readme-changes'
|\
| * badf5fd New README with new samples
* | 49a695c added samples file
|/
*   7726932 resolved merge conflict in README
|\
| * c7c0d3e added new comments (Collaborator)
* | 124be36 Added project starting date
|/
* d50fd4f README modified and Manual.txt created
* 976819f First commit

#we delete the branch once merged
$ git branch  –d  readme-changesg
```

9 Branch merge: resolving conflicts

- If two branches changed the same part of the same file, Git stops right before the merge commit. Conflicts must be resolved manually

- Conflict resolution: edit/stage/commit workflow

  - git status   shows which files need to be resolved
  - edit and fix the conflict
  - git add   the conflicting file
  - git commit   to generate the merge commit

9 Branches and remotes

- Remote branches are a special case of a local branch. In fact, a remote repository is also a hidden branch.

```
$ git branch -a
* master
  remotes/origin/master
```

A collaborator can develop a new branch

```
$ git checkout -b new-method
Switched to a new branch 'new-method‘
$ echo "New method file with 2 values" >> methods.md
$ git add .
$ git commit -m "added new methods file"
[new-method 82379ca] added new methods file
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 methods.md
```

9 Branches and remotes (example)

And pushed it into the central repository

```
$ git push origin new-methods   (or git push https://youraccount@github.com/youraccount/project_1.git new-method)
Counting objects: 15, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.26 KiB, done.
Total 15 (delta 2), reused 8 (delta 0)
To https://masenar@github.com/masenar/project_1.git
 * [new branch]      new-method -> new-method                 //collaborator has added new branch to CR

// We can see this new branch
$ git fetch origin
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 2 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/masenar/project_1
 * [new branch]      new-method -> origin/new-method      //we synchronize with our remote branches
```

9 Branches and remotes (example)

We can then create a local copy of that branch

```
$ git checkout -b new-method origin/new-method
Branch new-method set up to track remote branch new-method from origin.
Switched to a new branch 'new-method'
```

// Git has notified us that it's tracking this branch. I.e., this local branch knows which remote branch to push to and to pull from if we use such commands without arguments.//

```
$ ls
data  Manual.txt  methods.md  README        // we have the same copy of the branch as it is at the CR.
$ cat methods.md                            // we can work on this branch and push changes to the
New method file with 2 values               // remote repository or merged it with our master
                                            //  branch,…
```

10. Miscellaneous commands

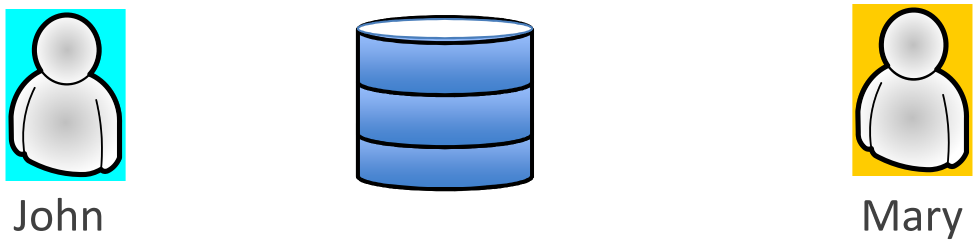
- Stashing your changes.

  git stash

  - Saves any working changes made since the last commit and restores repository to the HEAD version.
  - Handy to save partial progress before we apply an operation that is best performed with a clean working directory (i.e., pull or branching)
  - Changes can be applyed again with *git stash pop.*

11. Comparing workflows

Example of a cooperating workflow based on a centralized repository



1. **someone initializes the central repository**

```
ssh user@host git init --bare /path/to/repo.git
```

2. everybody clones the central repository

```
git clone ssh://user@host /path/to/repo.git
```

11. Comparing workflows

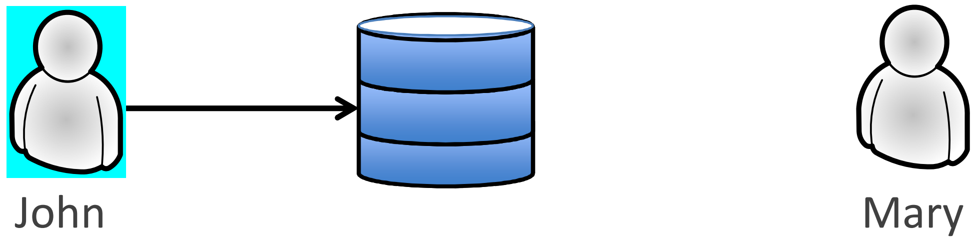
3. John works on his feature (editing, adding and commiting)

```
git status   # View the state of the repo
git add <some-file>   #Stage a file
git commit            # Commit a file </some-file>
```

Meanwhile, Mary is working on her own feature in her local repository

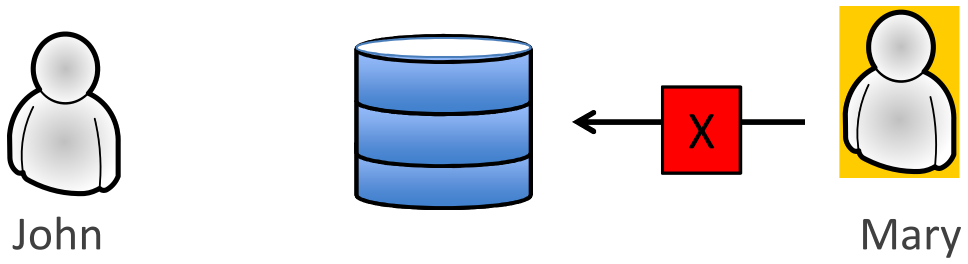
4. John publishes his feature

```
git push origin master
```



11. Comparing workflows

5. Mary tries to publish her feature



```
git push origin master
```

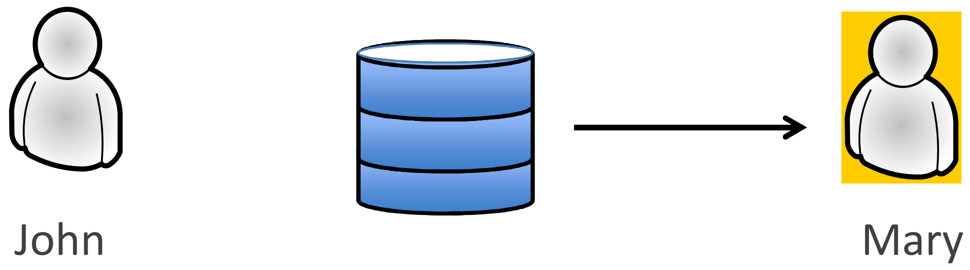
but, since her local history has diverged from the central repository, Git will refuse the request. This prevents Mary from overwriting official commits

```
error: failed to push some refs to '/path/to/repo.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')
hint: before pushing again.
hint: See the 'Note about fast-forwards' in 'git push –help' for details.
```

11. Comparing workflows

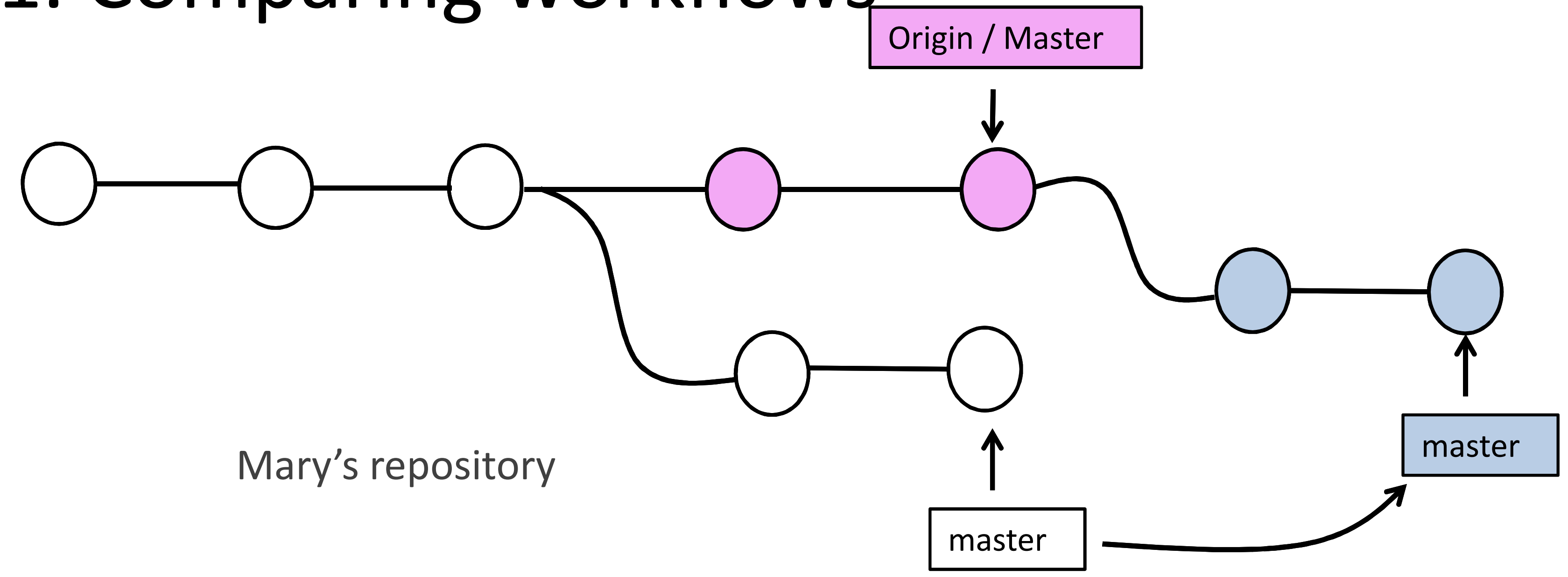
- **Mary rebases on top of John's commit(s)**

Mary needs to pull John's updates into her repository, integrate with them her local changes, and then try again



```
git pull --rebase origin master
```

11. Comparing workflows



The --rebase option tells Git to move all of Mary's commits to the tip of the master branch after synchronising it with the changes from the central repository

11. Comparing workflows

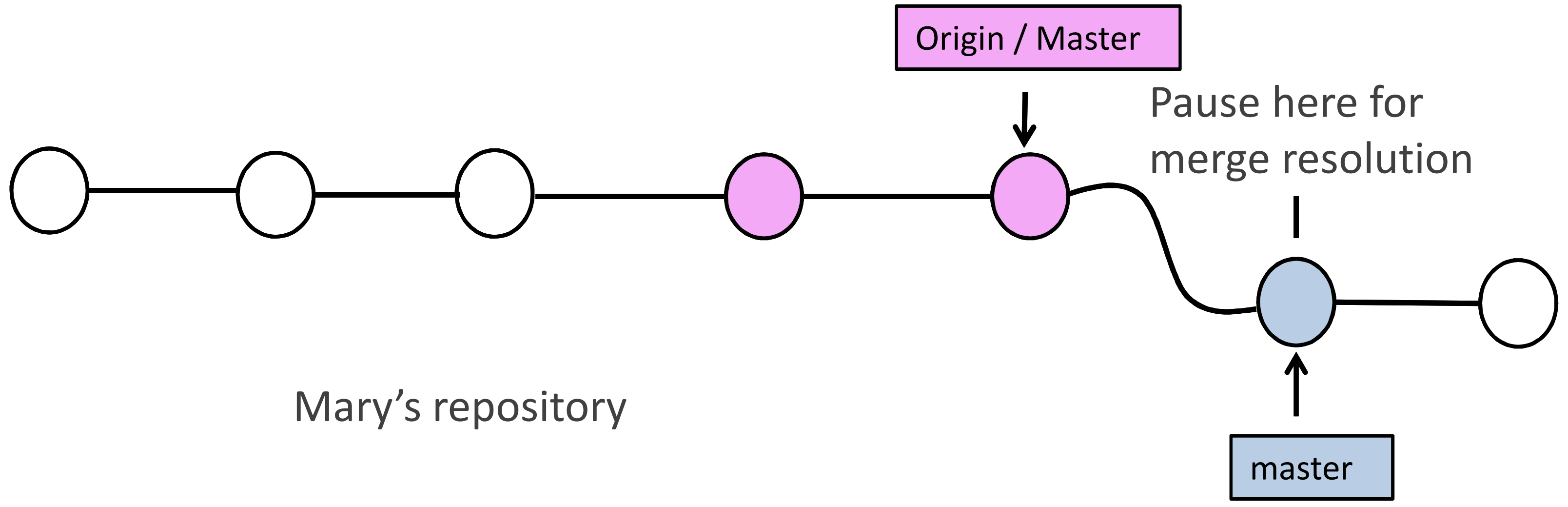
- **Mary resolves a merge conflict**

  Rebasing works by transferring each local commit to the updated branc one at a time. Conflicts are cached on a commit-by-commit basis (makes for a clean project history and makes much easier to figure out where bugs were introduced).

  In case of conflict, Git will pause the rebase at the current commit and output the following message, along with some relevant instructions:

```
CONFLICT (content): Merge conflict in <some-file>
```

11. Comparing workflows



- git status shows where the problem is

```
# Unmerged paths: # (use "git reset HEAD <some-file>..." to unstage)
# (use "git add/rm <some-file>..." as appropriate to mark resolution)
#
 # both modified: <some-file>
```

11. Comparing workflows

Mary edits the file(s) to her liking and she can stage the file(s) as usual and git rebase do the rest.

```
git add <some-file>
git rebase –continue
```

Git will move on to the next commit and repeat the process for any other commits that generat conflicts.

```
git rebase --abort
```

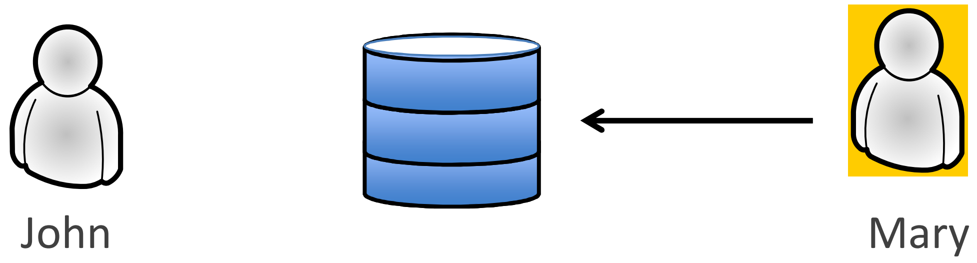
can be used to stop rebase and go right back to where git pull --rebase was started.

11. Comparing workflows

- **Mary succesfully publishes her feature.**

After she's done synchronizing with the central repository, Mary will be able to publish her changes succesfully.

```
git push origin master
```



References

- “Pro Git”, Scott Chacon, Apress, 2014.
- “Git for Scientists (chapter 5) ”, in Bioinformatics Data Skills, Vince Buffalo, O’Reilly Media Inc., 2015.
- http://git-scm.com/docs
- https://www.atlassian.com/git/tutorials
- http://www.vogella.com/tutorials/Git/article.html
- https://github.com/
- https://bitbucket.org/