ESCOLA D'ENGINYERIA

Parallel and Distributed Computing (calculation) Systems

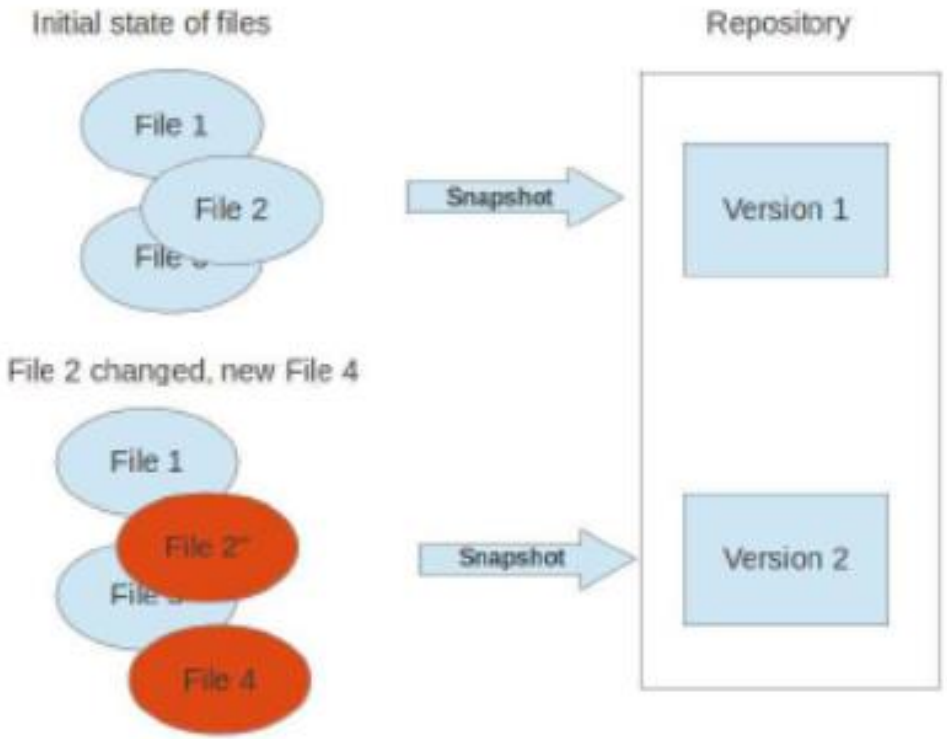
Version control systems

R. Suppi (Remo.Suppi@uab.cat)

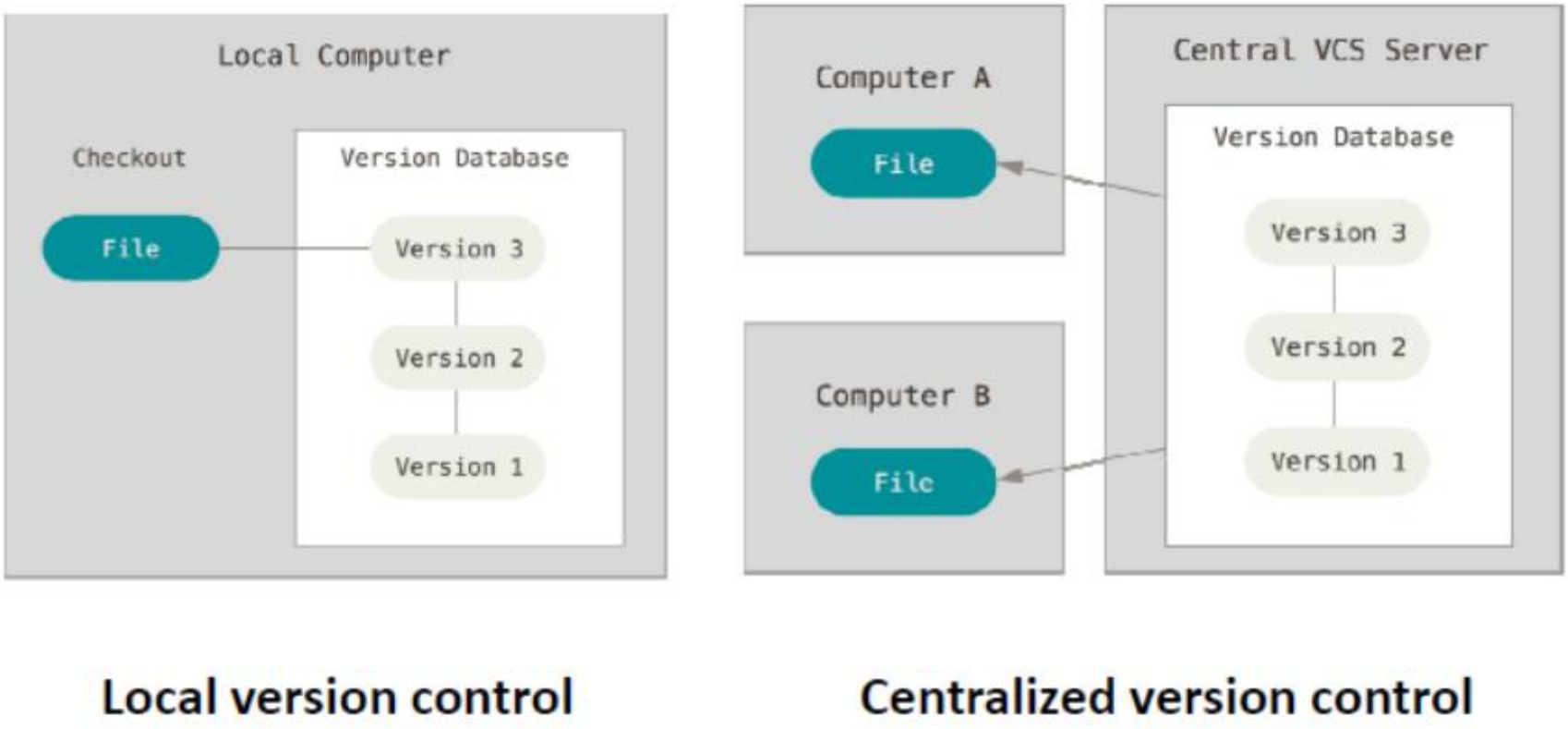
What is a version control system?

Concept

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



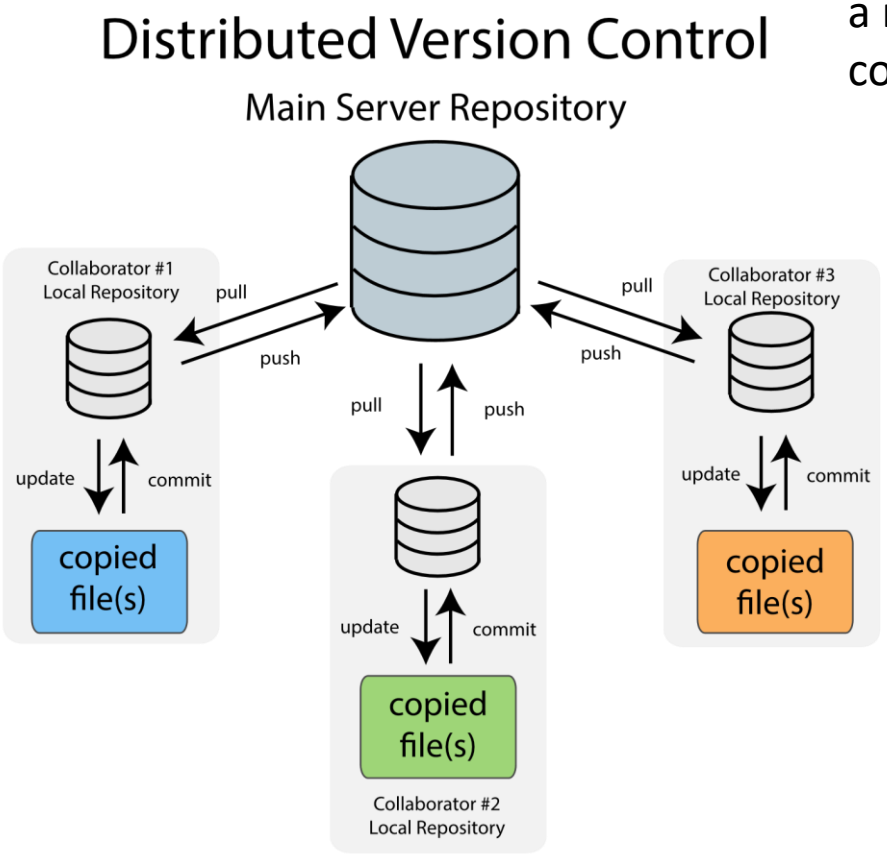
Local vs. Centralized?



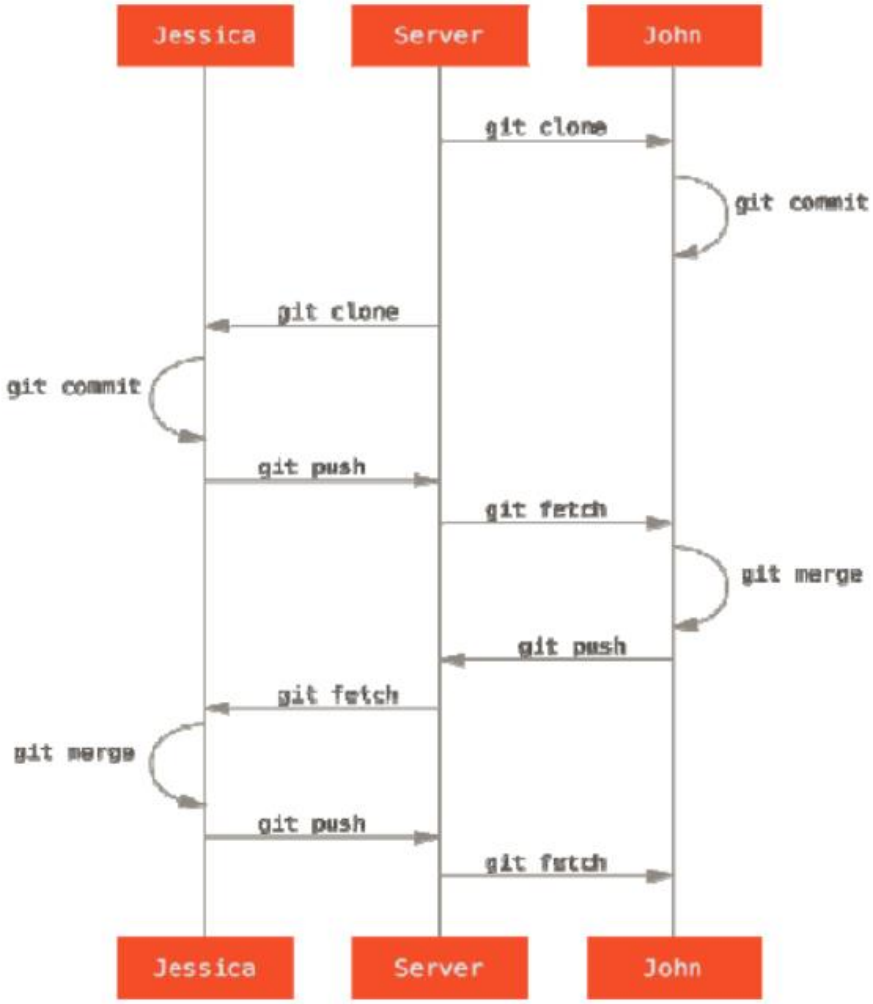
Local version control **Centralized version control**

Distributed?

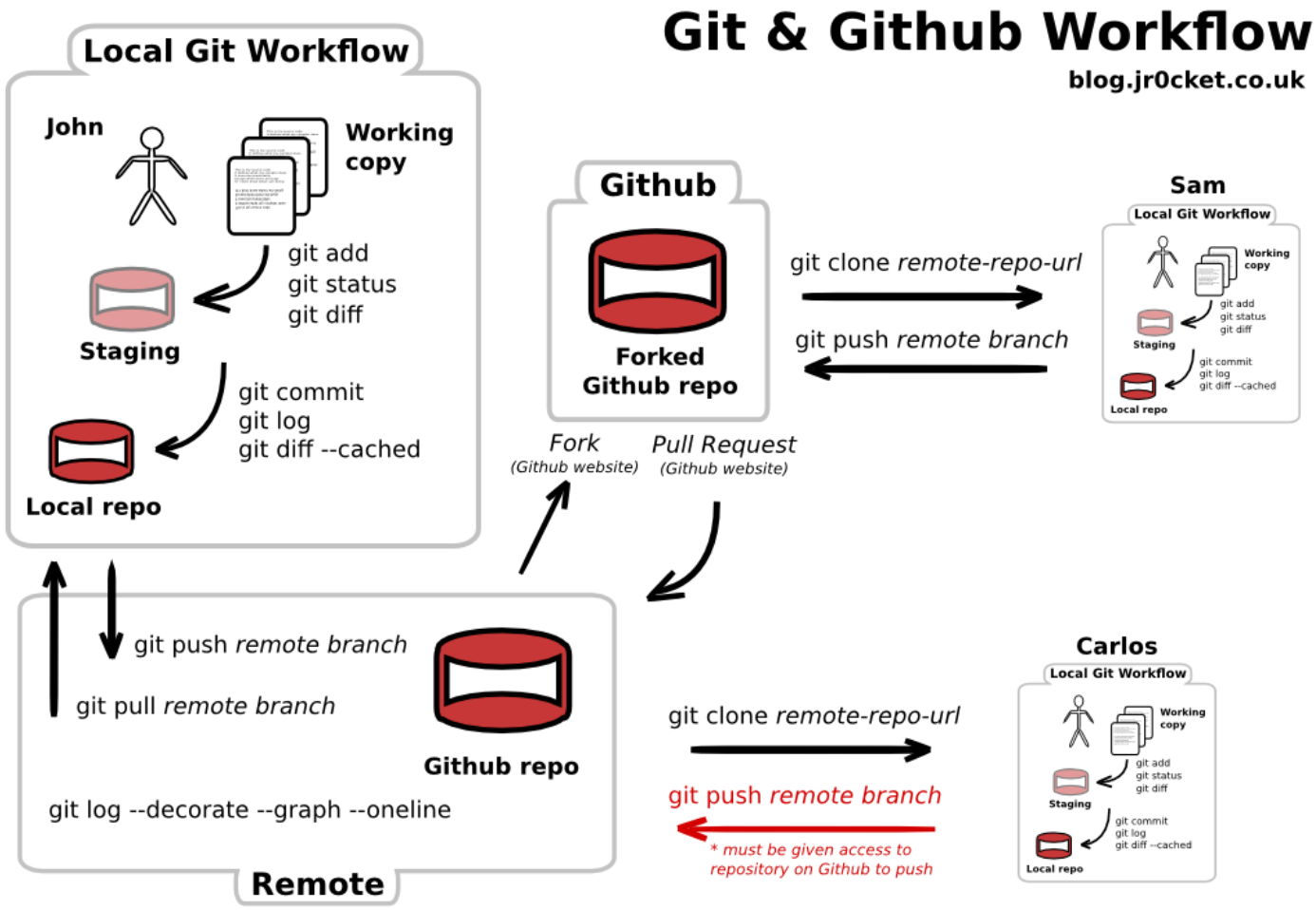
- A distributed version control system does not necessarily have a central server which stores the data.
- Typically there is a central server for keeping a repository but each cloned repository is a full copy of this repository.



General Sequence of two users Git Workflow



General Sequence of Git-Github Workflow



Git for scientists

- Git originally intended to work with large software projects.
- Complicated scientific 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 isolation by rolling back to past versions of the project.

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).

  Linux: gitk, git gui, SmartGit, Eclipse EGit
  Windows: Git forWindows, SmartGit , SourceTree, GitEye

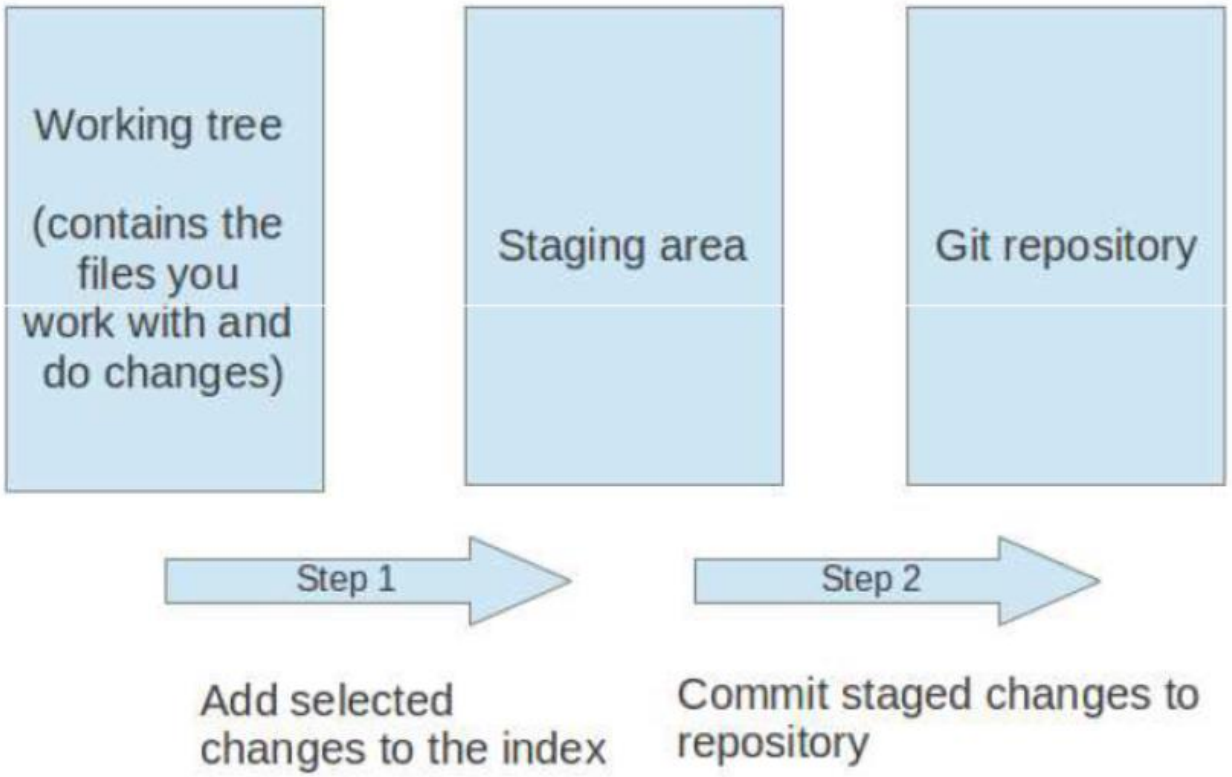
- A **repository** contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository.
- A **branch** is a named pointer to a commit. When you commit your changes into a repository this creates a new commit object in the Git repository.
- A **tag** points to a commit which uniquely identifies a version of the Git repository.

Hosting

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.

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.

Basic Operation



Setting up a repository

Installation as usual software (eg. *apt-get install git* in Ubuntu).

Some initial configuration (.gitconfig file)

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

.gitignore file: to ignore certain files and directories

note: in scientific projects, large files (such as FASTA or FASTQ used in Genomic studies) should be ignored and not be included in the repository.

Setting up a repository

Creation of a repository

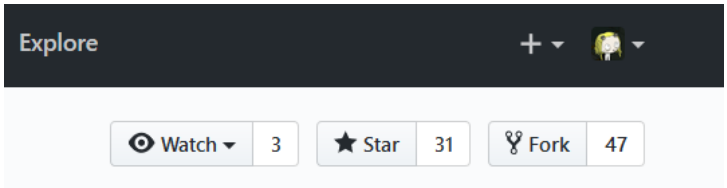
git init <directory> (developers local repository)
git init -- bare <directory> (central repository)

Cloning a repository

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

Github repository: https://github.com/banago/simple-php-website

Fork to your account:



https://github.com/rsuppi/simple-php-website.git

Now, let's clone simple-php-website into your local machine
$ git clone https://github.com/rsuppi/simple-php-website.git
(or) $git clone git:// github.com/rsuppi/simple-php-website.git
check the contents of simple-php-website directory (using ls).

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.dat
```

Saving changes

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

  git add <file>

  git add <directory>

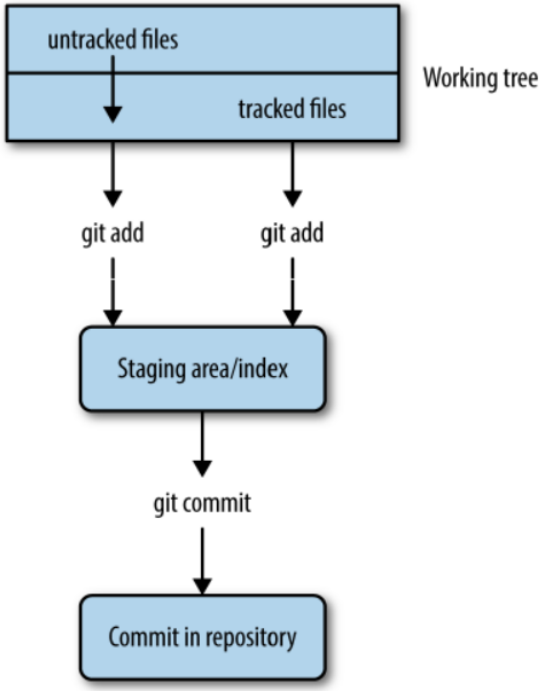
- Committing the staged snapshot to the project history

  git commit

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

- List of files staged, unstaged and untracked

  git status



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

--- - 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

$ **git commit –m “My first commit”**
[master (root-commit) 976819f] First commit
Committer: rsuppi <rsuppi@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

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

$ **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: rsuppi rsuppi@aolin21.uab.es
……..
2 files changed, 2 insertions(+), 0 deletions(-)
create mode 100644 Manual.txt

Example 2

```
$ echo “Project started 10-19-2017” >> 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

$ git log               ------ shows the history of commits
$ git log --graph --pretty=short
```

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*)

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

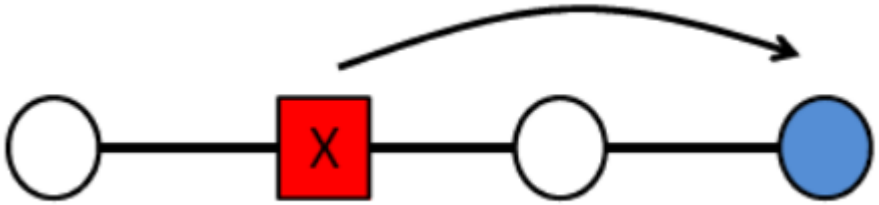
$ git checkout -- README // "--" avoids potential confusion with a branch named README
$ more README
Project created by Paul
I have added a second line at README

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

Viewing old commits and undoing changes

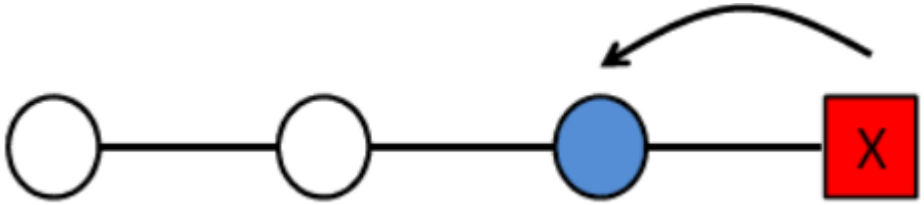
Undoing a commited snapshot (safe way to undo changes). Generates a new commit that undoes all of the changes introduced in <commit>, then apply it to the current branch.

git revert <commit>



Removing commited snapshots (dangerous way to undo changes)

git reset <commit>



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.

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*

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

 # clone online repository

git clone git://github.com/vogella/gitbook.git

 # clone online repository

git clone ssh://git@github.com/vogella/gitbook.git *(read-write; requires valid SSH account)*

 # the following will clone via HTTP

git clone http://github.com/vogella/gitbook.git (normally read-only)

Working with remote repositories

Importing commits from a remote repository to the local repo.

git fetch <remote> fetch all branches

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

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)

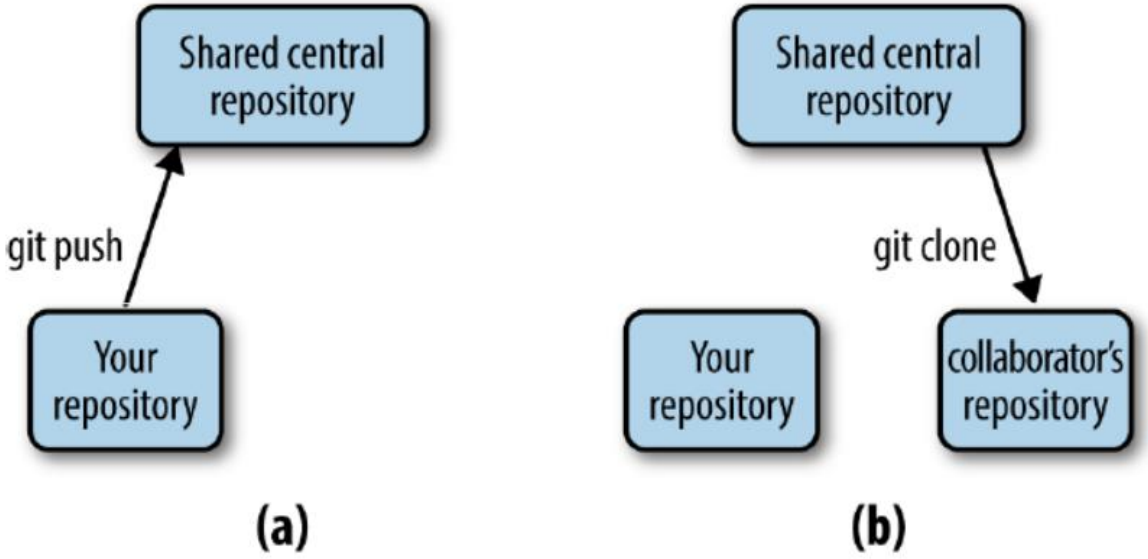
– Push your project's initial commit

– 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

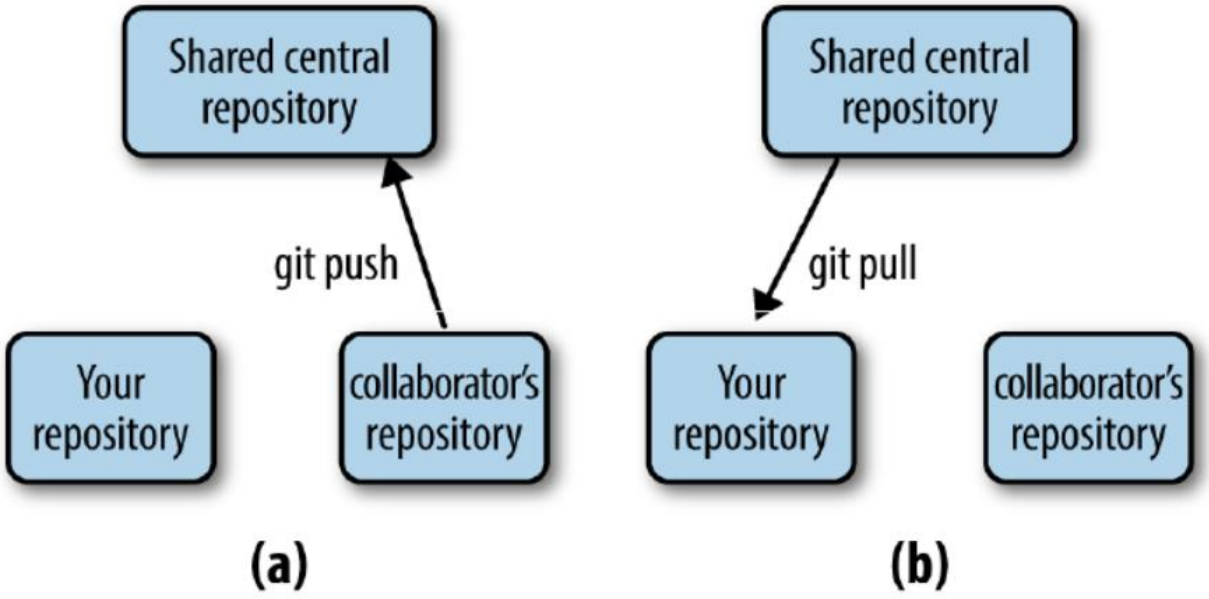
Working with remote repositories



(a) Creation of a new shared CR

(b) Collaborator retrieves project

Working with remote repositories



(a) Collaborator pushes changes to CR

(b) Retrieving changes made by collaborator

Working with remote repositories

Transferring commits from local repository to a remote repository. git push used to publish local changes to central 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

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)

Creating a shared central repository with GitHub

$ 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

Creating a shared central repository with GitHub

--- 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: rsuppi rsuppi@aolin21.uab.es ----- (should be collaborator's username in a real scenario)
.....
1 files changed, 1 insertions(+), 0 deletions(-)

Creating a shared central repository with GitHub

$ git push https://rsuppi@github.com/rsuppi/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://rsuppi@github.com/rsuppi/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)

If you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with
git remote add origin <server>
Now you are able to push your changes to the selected remote server

$ 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

Creating a shared central repository with GitHub

---- suppose we have already made same changes to our local repository
$ echo “Project started 10-19-2017” >> 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 <rsuppi@aolin21.uab.es>

----- we try to push it to CR
$ git push origin master
Password:
To https://rsuppi@github.com/rsuppi/project_1
! [rejected] master -> master (non-fast-forward)
error: failed to push some refs to 'https://rsuppi@github.com/rsuppi/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

Creating a shared central repository with GitHub

---- We pull from the CR to get modified files

$ git pull origin master

```
Password:
From https://github.com/rsuppi/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")
```

Creating a shared central repository with GitHub

$ 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

Creating a shared central repository with GitHub

$ 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: rsuppi <rsuppi@aolin21.uab.es>

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

Additional features:

- Branching & Merging
- Comparing Workflows

References:

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/
http://rogerdudler.github.io/git-guide/

Answers & comments