Version Control for Computational Economists: An Introduction

Jake C. Torcasso

April 4, 2014

Starting Point

- A collection of files on your computer
- Changes to files and new files over time
- Interested in preserving the history of these changes

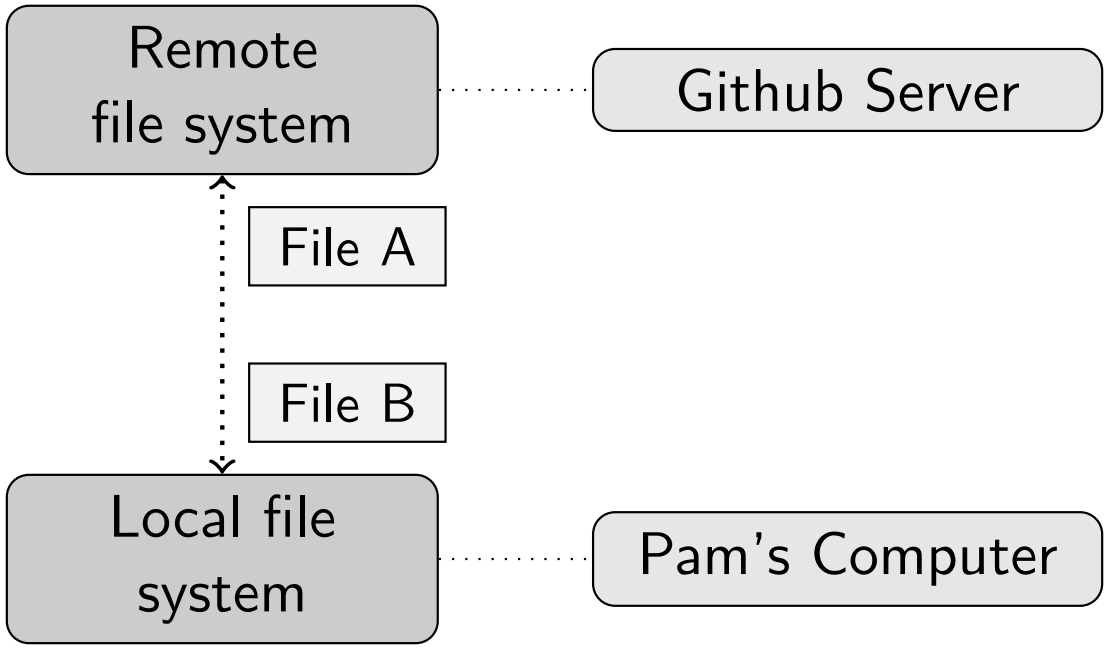
In one sentence...

"Version Control is a system that records changes to a file or set of files over time so that you can recall specific versions later" (Chacon and Hamano, 2009).

Version Control Evolving

- Middle Ages - Copying the Bible
  - Each version was handwritten
  - Used margins for corrections
  - Induced regional heterogeneity
- The modern Bible scribe
  - Copy/paste versions to an archive
  - Include a readme
- Post-modern methods of Version Control
  - Version Control Systems
    - Localized (rcs)
    - Centralized (CVS, Subversion, Perforce)
    - Distributional (Git, Mercurial, Bazaar, Darcs)

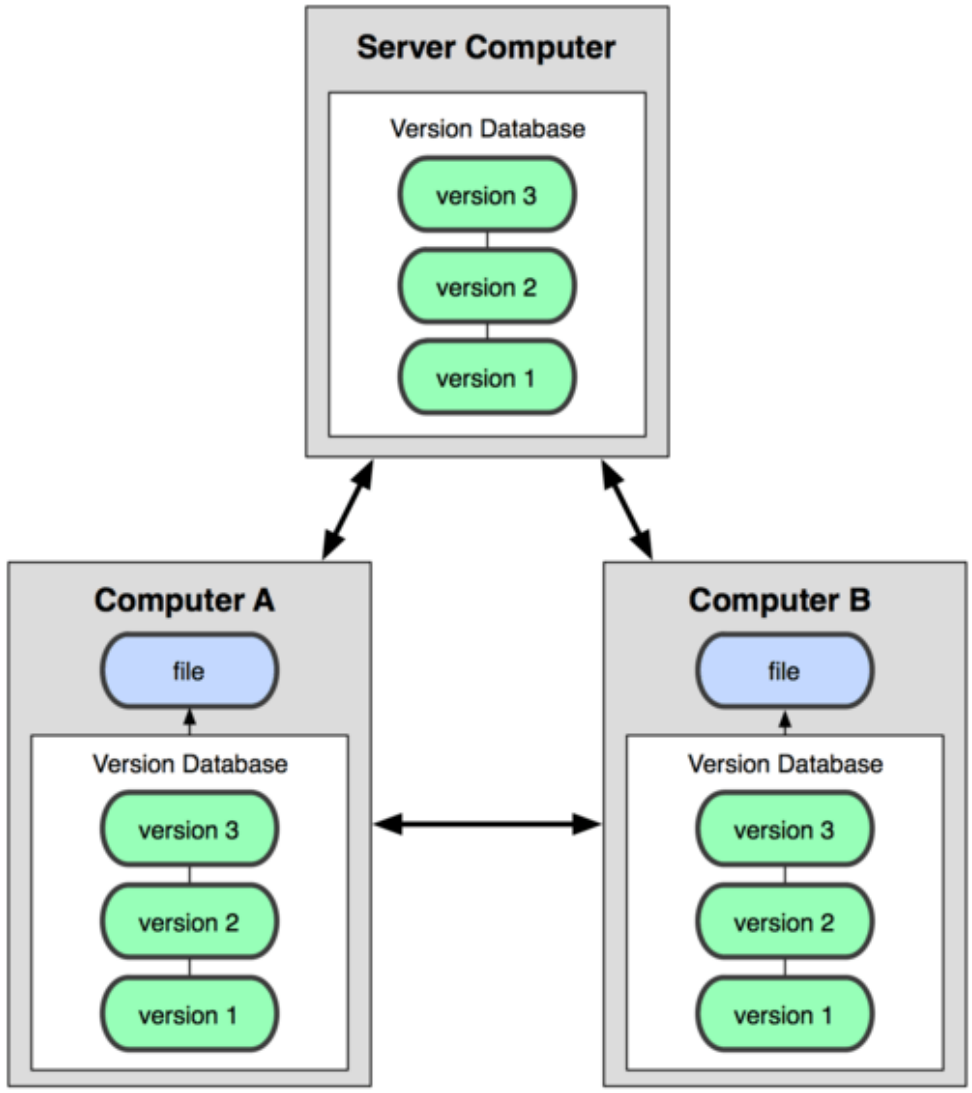
Concepts of Version Control Systems



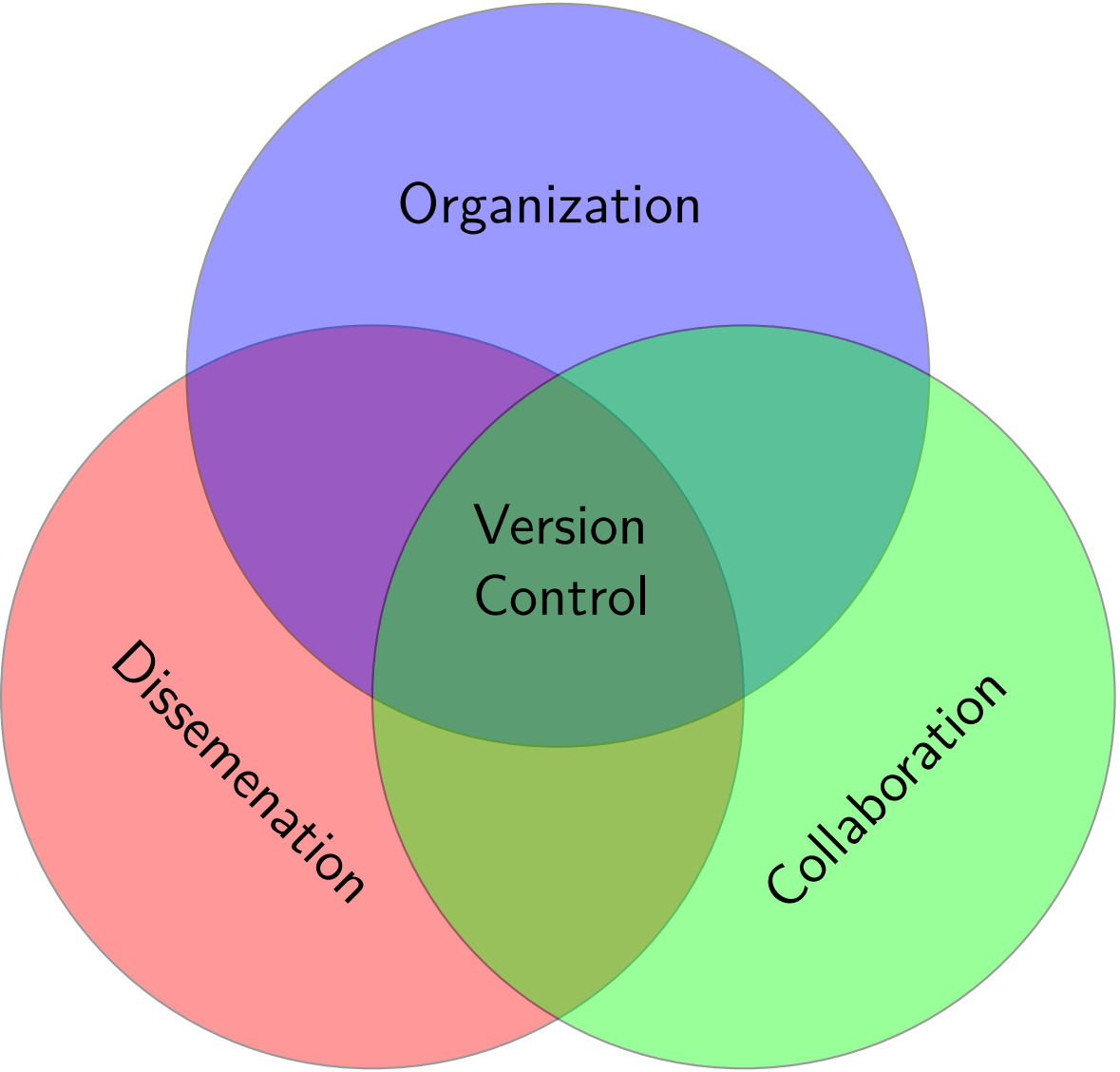
Distributional Version Control Systems

- Network of repository copies (mirrors)
- Identical, full copies of the data

DVCS Structure



Version Control in Economic Research



- Dissemenation
  - Availability
  - Reproducibility
  - Transparency
  - Extendability
- Organization
  - Contemporaneity
  - Workflow
  - Project Management
  - Progress tracking
  - Security
- Collaboration
  - Visibility
  - Communication
  - Coordination

Dissemenation

- Self-contained source code
- Online visualization and availability
- Seamless integration with existing knowledge...
  - Reduces burden to reproduce work
  - Provides immediate stepping stone for future work
  - Meaning more scientific progress!
- Facilitates review of scientific work
  - Too often overlooked and under-emphasized

You'll believe me if you go on Github.com.

Organization

- Always stay up-to-date
- Explore alternative workflows–diverge and converge
- Chill pill–at ease with experimentation
- Retain an (annotated) historical record of your work
- Manage access rights

Collaboration

- Increase oversight over project contributors
  - Check logs for progress updates
  - Set milestones and tag important project states
- Quickly point-out issues (bugs)
- Resolve file conflicts
- Increase foresight
- At-ease with the newbies
  - Erase mistakes
- Non-linear project workflows
- Easily merge work from others

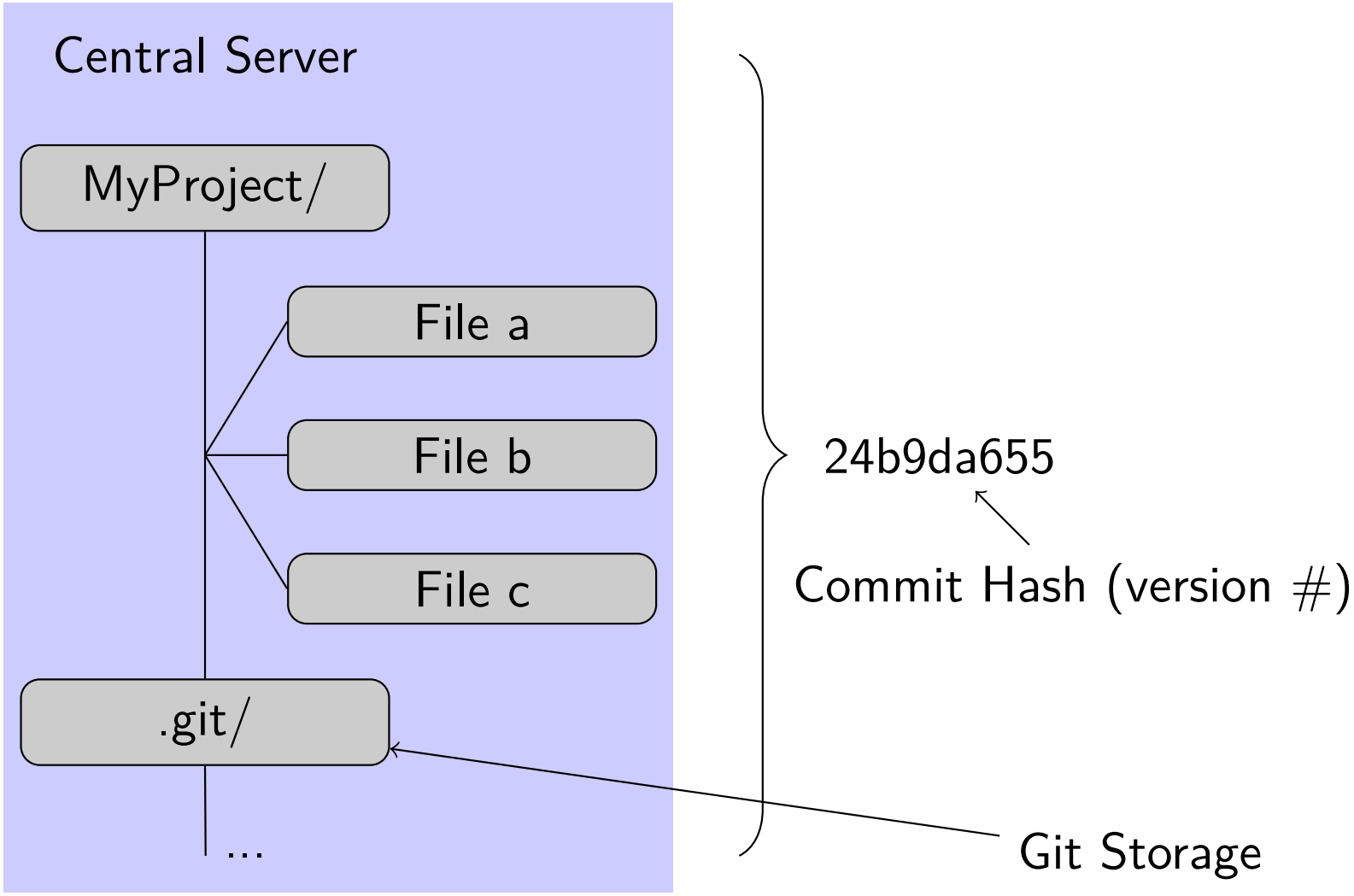
Git

Introducing Git

Git is a distributional version control system most notably used by Github, the web-based hosting service for software development projects. Checkout a good Git book here.

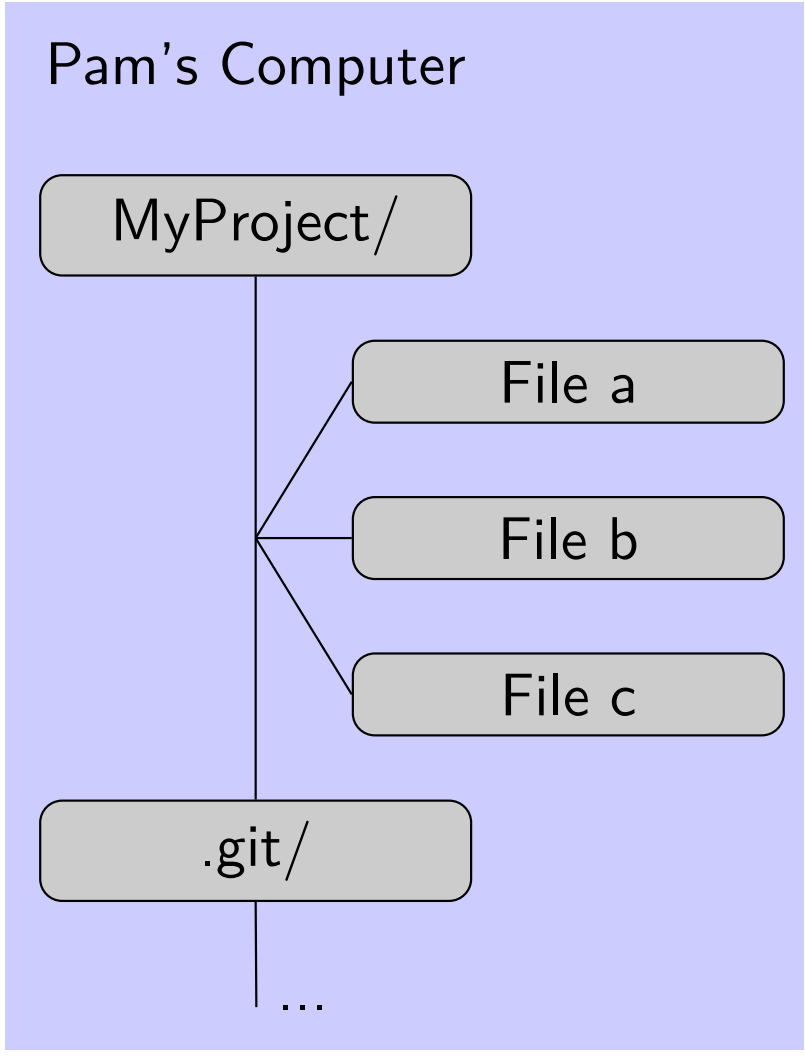
Introducing Git

First consider a git server, which is nothing but a computer with the following file structure.



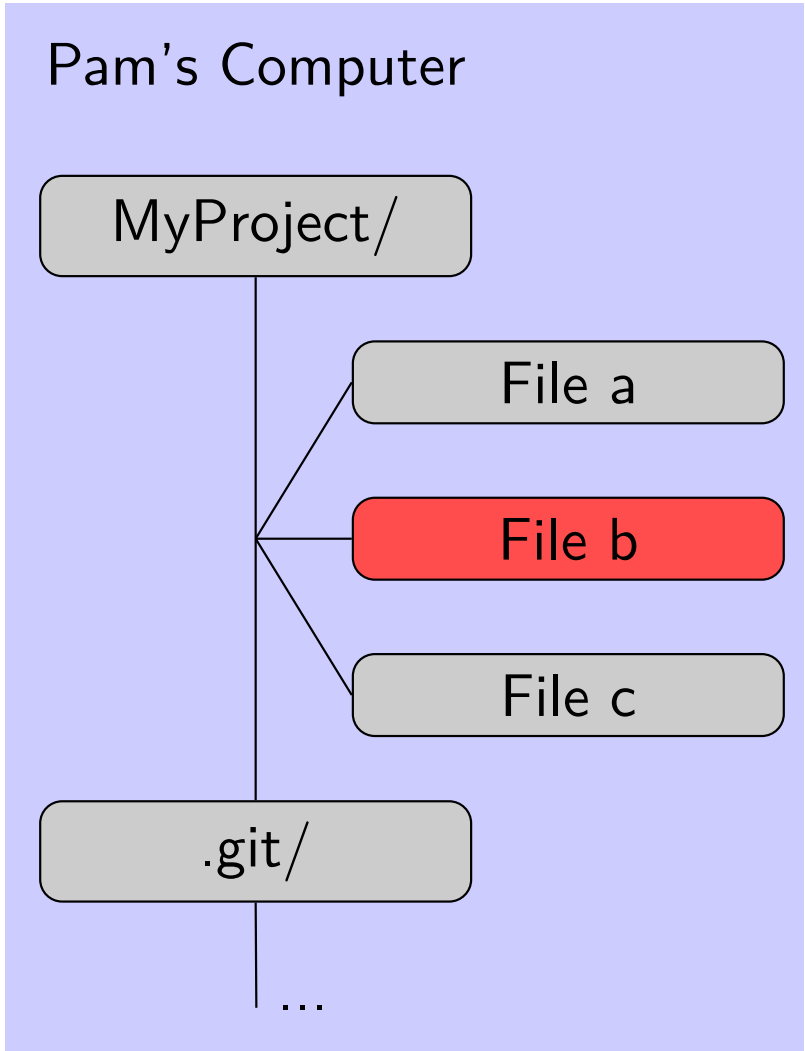
Introducing Git

When Pam clones this repository to her computer, she sees:



Introducing Git

When Pam changes “File b”, she merely changes her “working directory”. Git will recognize the change, but won’t record it.



Introducing Git

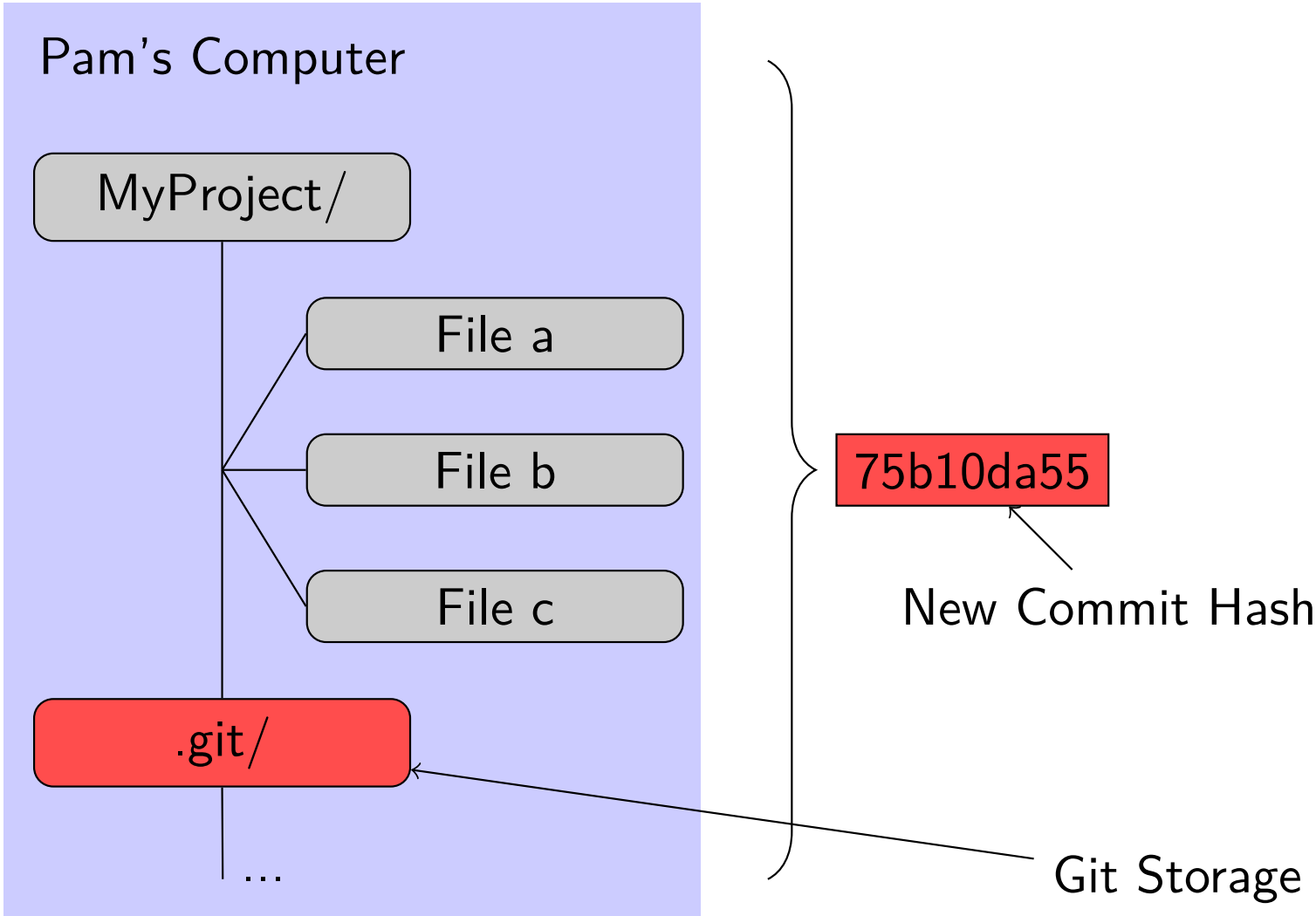
To record the change she performs two commands:

1. `$ git add ‘‘File a’’`
2. `$ git commit -m ‘‘I have changed File a.’’`

The second command generates a commit and a corresponding commit message. A commit is like a “snapshot”; it records the current state of your files. Read more on commits here.

Introducing Git

Now Pam's local repository is at a future state, recorded as a new commit hash. The commit information is stored in the git directory.



Introducing Git

Using `$ git status`, we get the following output:

```
$ git status
# on branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
nothing to commit (working directory clean)
```

The git directory stores the commit history:

... → 24b9da655 → 75b10da55

Using `$ git status` told Git to compare the current state with the last known state directly from 'origin/master'.

Introducing Git

To see the last 2 commits we may do the following:

```
$ git log -2
commit 75b10da55
Author: Pam <pam@usa.com>
Date:   Mon Mar 24 17:28:17 2014 -0500

    I have changed File a

commit 24b9da655
Author: David <david@milkandcheese.com>
Date:   Tue Mar 13 12:33:16 2014 -0500

    Included this month's cow deaths in File c
```

Introducing Git

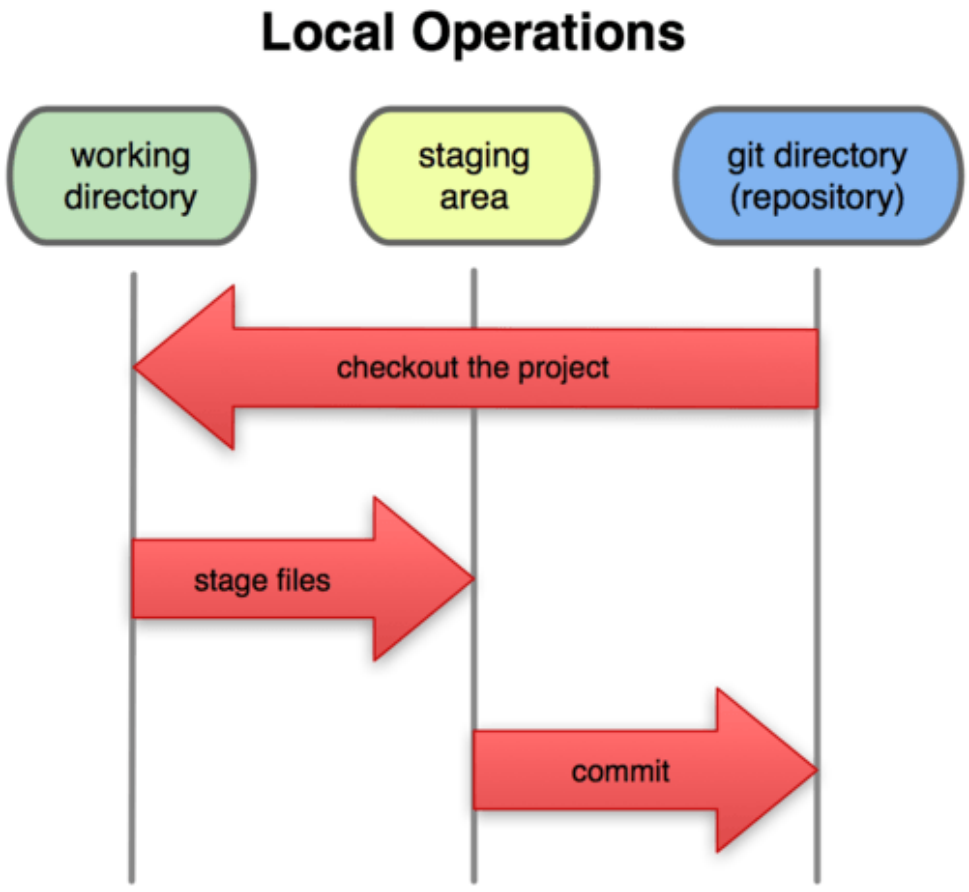
To introduce her changes to the Central Server, Pam has to push her changes.

```
$ git push origin master
```

The Central Server has been updated with Pam's changes.

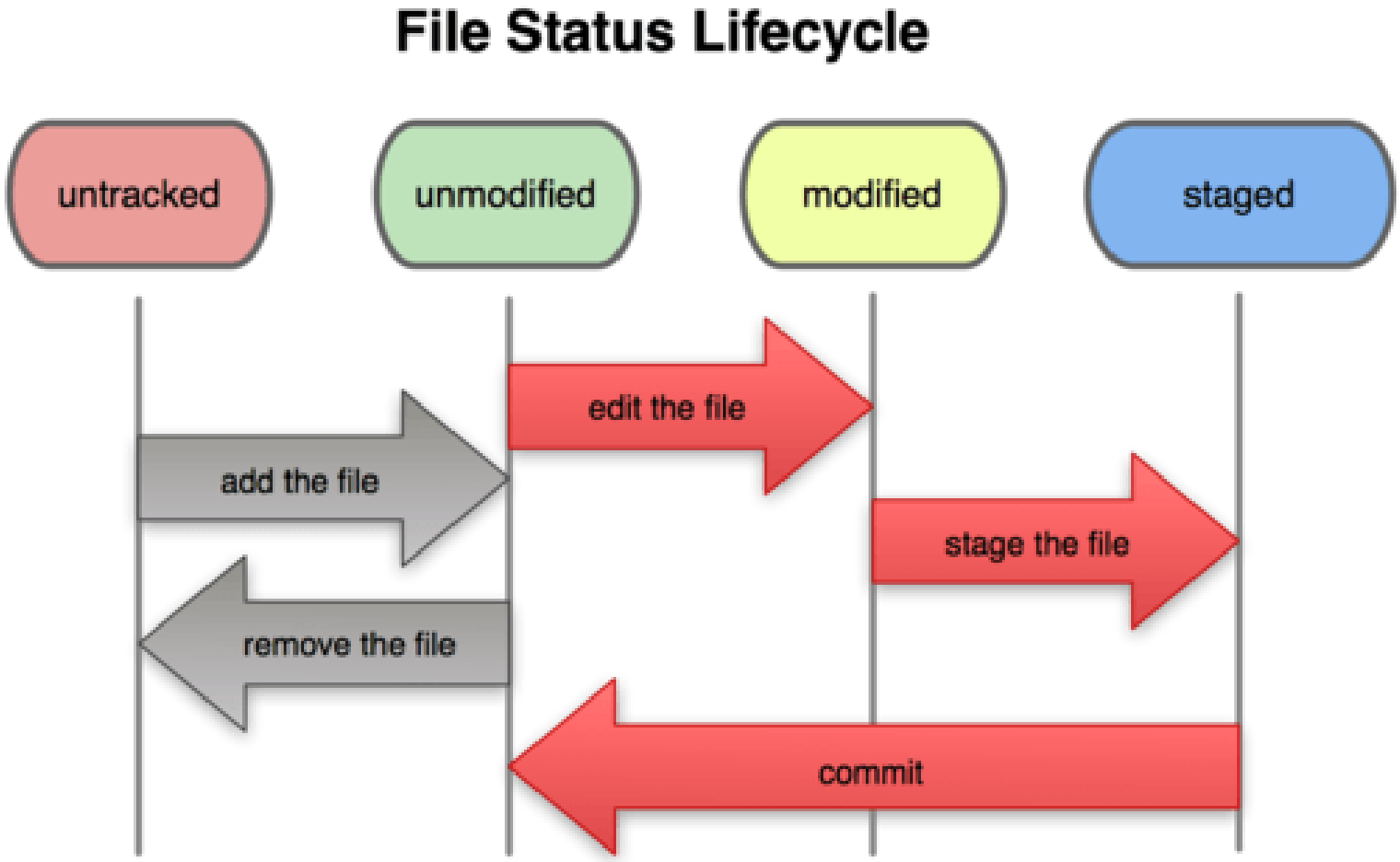
Git Concepts

Now that you have been introduced to Git, let's clarify some of the concepts you have encountered.



Git Concepts

We have also seen the various ways Git recognizes and records information about files.



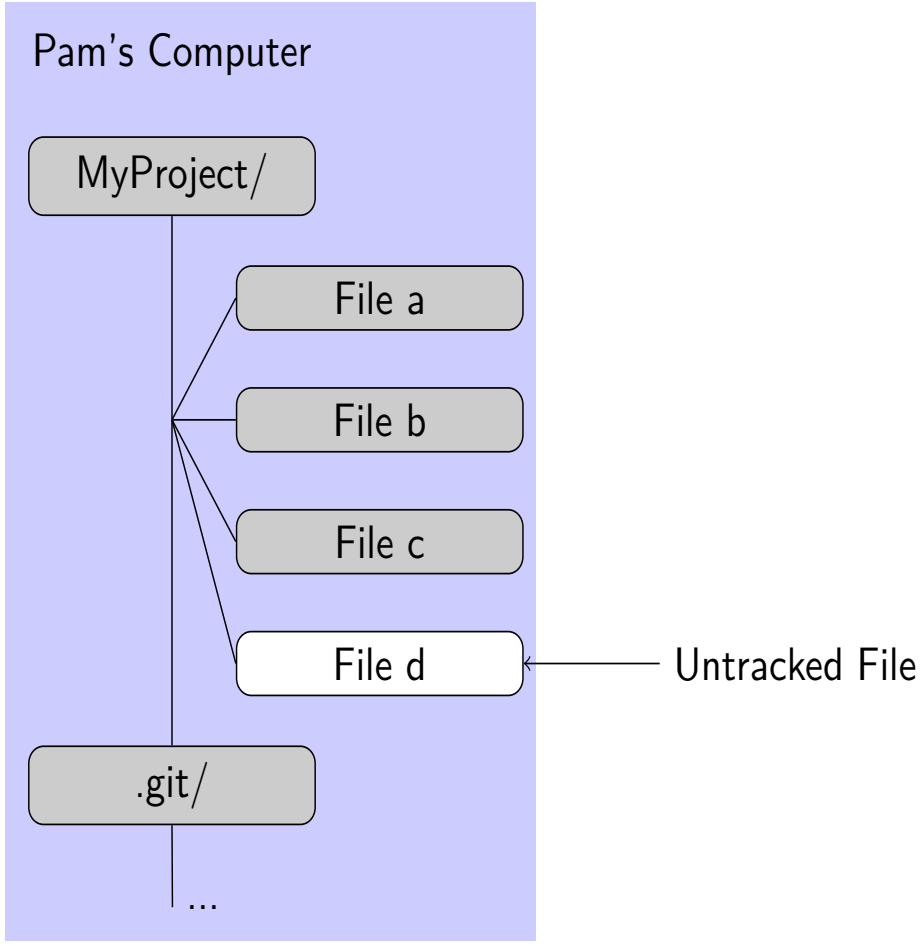
Git Concepts

Tracking:

- Git will only track files you tell it to track
- Only tracked files have a commit history, enabling:
  - updates to remote repositories
  - reverting changes

Git Concepts

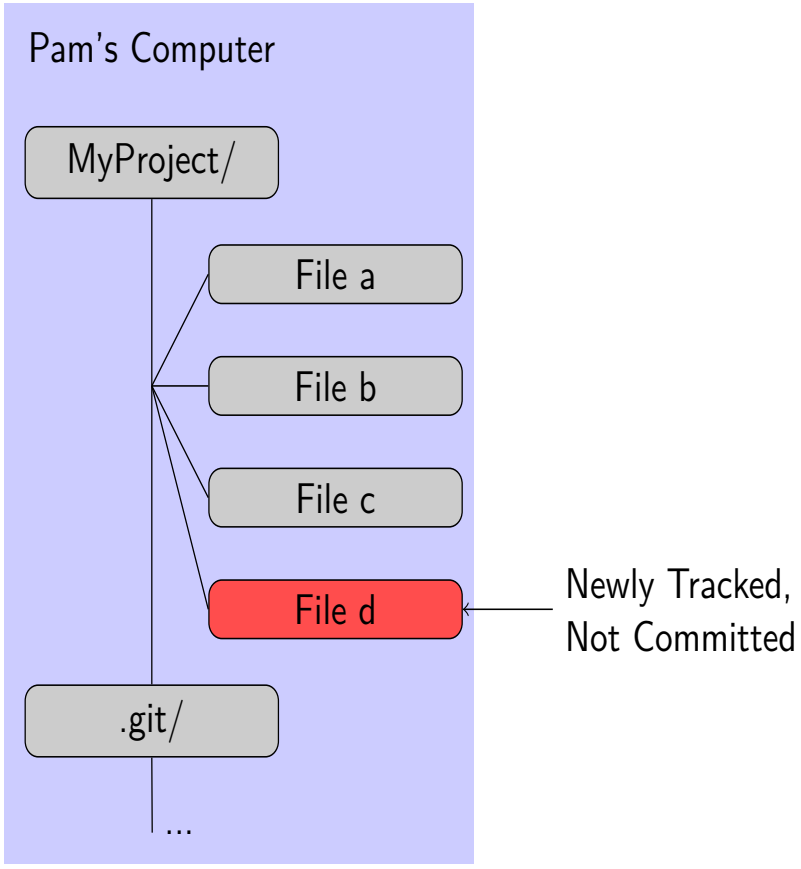
Let's see how Pam begins tracking "File d", which she just created and added to her project. Her working directory looks like this:



Git Concepts

Pam opens terminal and issues the following command:

```
$ git add ‘‘File d’’
```



Git Concepts

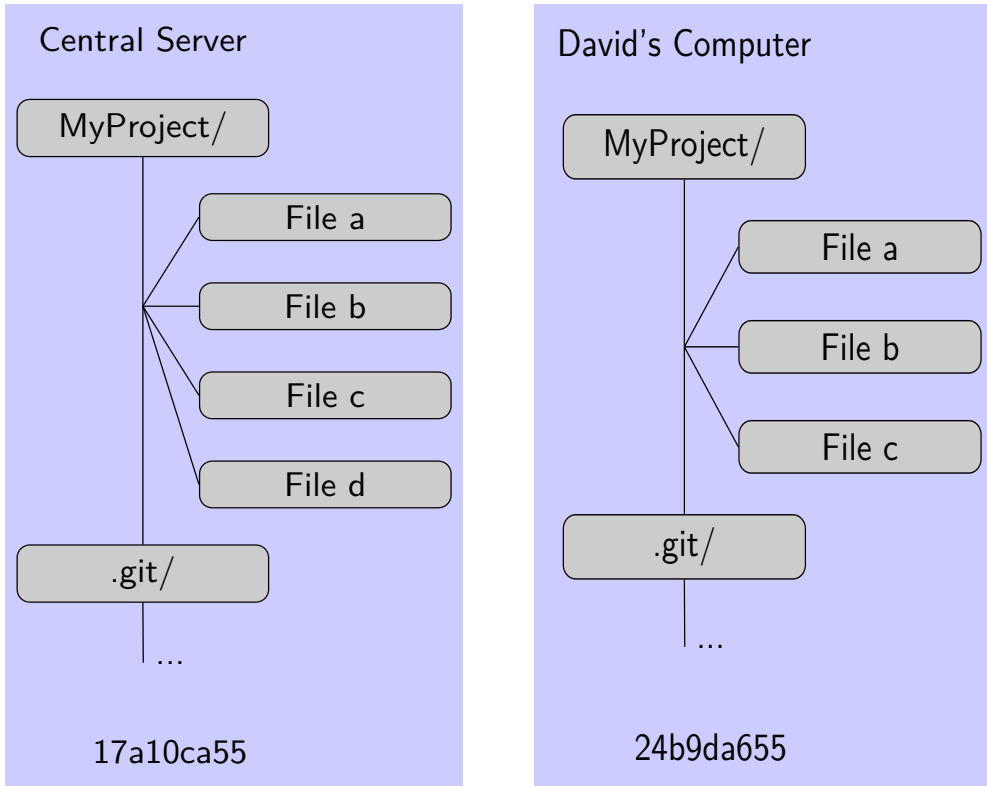
Pam pushes her changes to the remote Central Server.

```
$ git commit -m "Added File d, contains info on fat%."
$ git push origin master
```

Collaboration with Git

Git Concepts

David wishes to update his local files with the most recent version from the Central Server (i.e. fast-forwarding to Pam's commit.)



Commit History:

... → 24b9da655 → 75b10da55 → 17a10ca55

Git Concepts

When David issues the command

```
$ git pull origin
```

the changes upstream are fetched from the Central Server and merged with the files in his working directory.

Branching

Git Concepts

Up until now, we have glossed over one very important feature of Git: branching.

But we have learned two concepts: the commit and git repository.

Git Concepts

A Git branch is just a **pointer to a specific commit.**

- allows for non-linear workflows and simultaneous channels of development.
- aids the implemention new features.

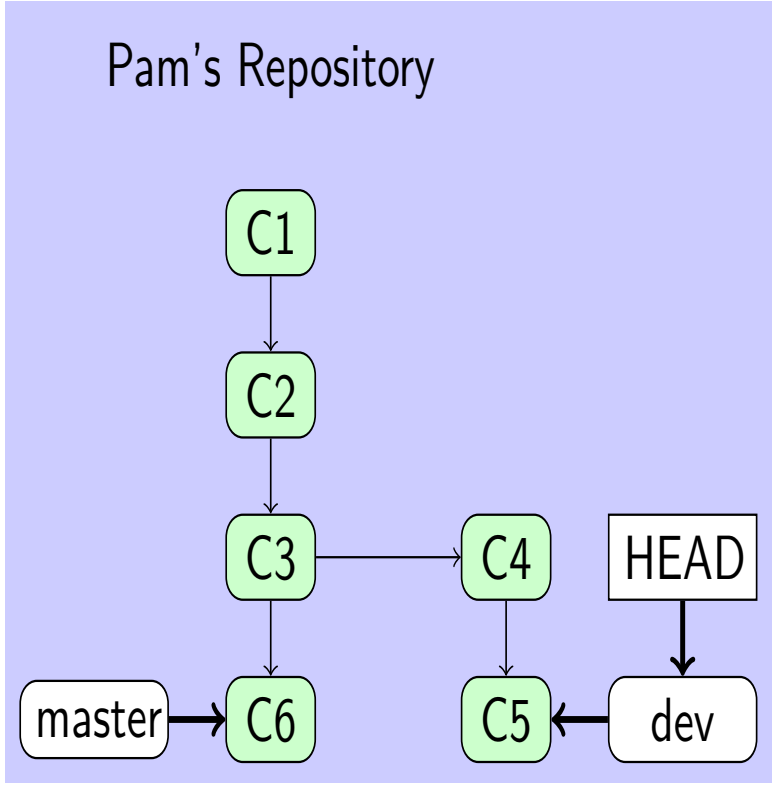
Git Concepts

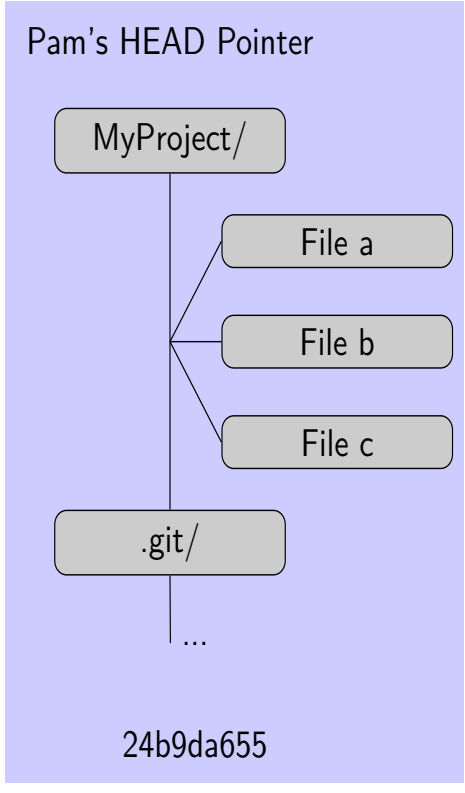
An economist might use branching to:

- Attempt a new identification strategy
- Quickly revert to a previous set of results
- Experiment with new numerical software

Git Concepts

Git repositories, commits and branches all describe a location in Gitland.





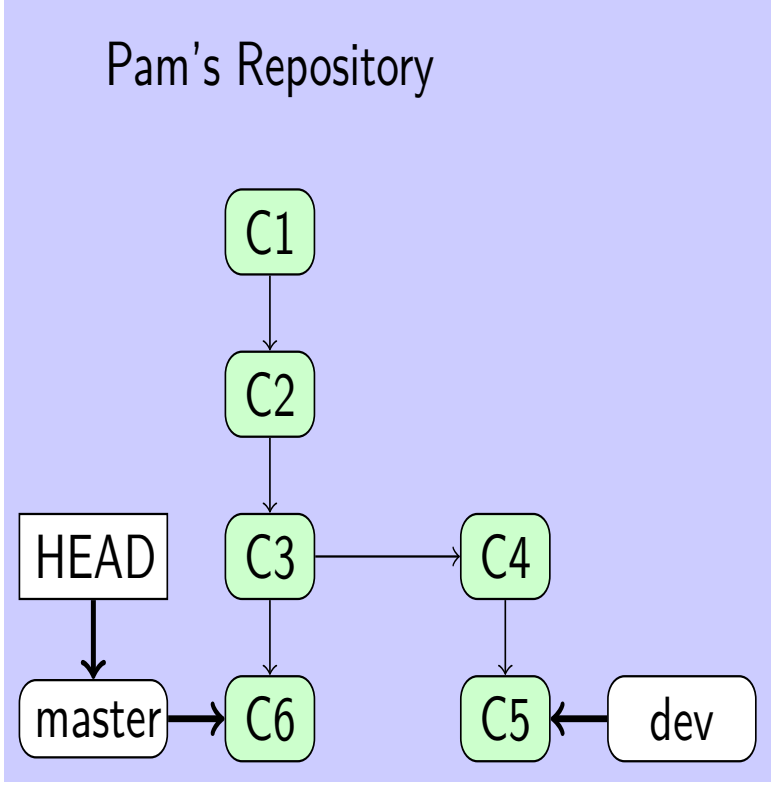
Git Concepts

HEAD is a special pointer which always points to the current focal branch. master and dev are branches, which merely point to a particular commit. Each commit is a saved state, or snapshot of your project *as a whole*.

Git Concepts

Navigate Gitland by changing the location of the HEAD pointer.
You can checkout a branch:

```
$ git checkout master
```



Merging

Git Concepts

We will not get into the details of merging, but we can explore one example. Let's have Pam merge the dev branch into master.

```
$ git merge dev
```

Git Concepts

If the `master` and `dev` branches did not modify the same file, the merge should go smoothly, producing an automatic merge commit. Otherwise, Pam has to modify the conflicted file(s) and then manually commit.

Git Concepts

Let's say Pam has a merge conflict. The conflicted file looks like this in the two different branches.

- dev

```
places = {'Mexico':'Spanish', 'United States':'English',
    'Brazil':'Portuguese'}

for key in places:
        print key, places[key]

for i in [1,2,3]:
        print i
```

- master

```
places = {'Mexico':'Spanish', 'United States':'English',
    'Brazil':'Portuguese'}

for key in places:
        print key, places[key]

for i in [4,5,6]:
    print i
```

Git Concepts

After attempting the merge, Git forces Pam to resolve all merge conflicts. Git modifies the file in her working directory to highlight the conflicting portions of the file.

```
places = {'Mexico':'Spanish', 'United States':'English',
        'Brazil':'Portuguese'}

for key in places:
        print key, places[key]

<<<<<<< HEAD
for i in [4,5,6]:
=======
for i in [1,2,3]:
>>>>>>> new
        print i
```

`<<<<<<<< HEAD` signals the version of your current branch and `>>>>>>>> new` that of the branch you attempting to merge into your current branch.

Git Concepts

Pam resolves the conflict by editing the file.

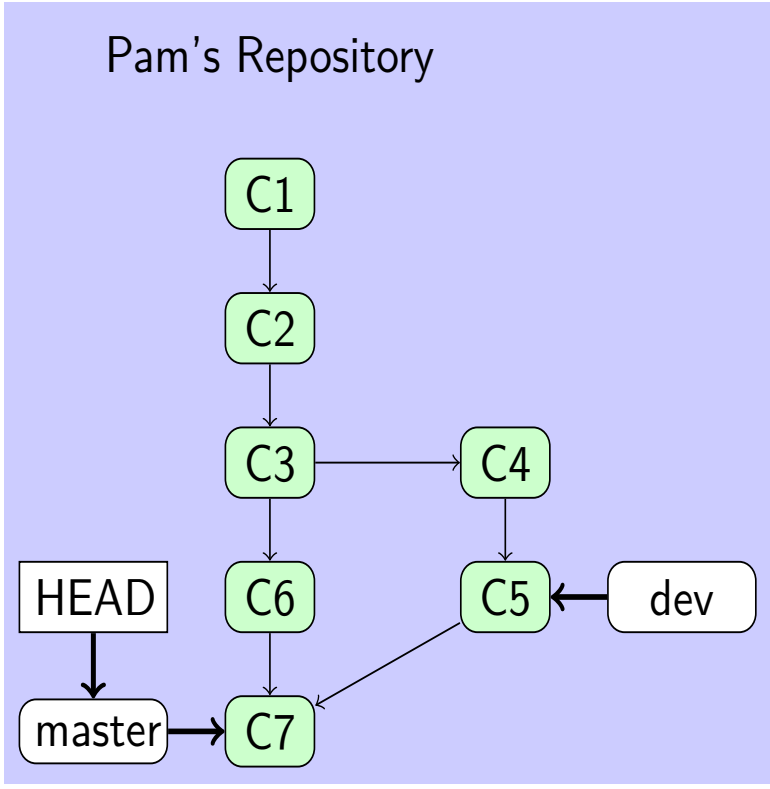
```
places = {'Mexico':'Spanish', 'United States':'English',
        'Brazil':'Portuguese'}

for key in places:
        print key, places[key]

for i in [1,2,3,4,5,6]:
        print i
```

Then she commits again.

```
$ git add filea
$ git commit -m "Resolved conflict, iterating through long list"
```

Git Concepts

After the merge is complete, Pam's commit history in her local repository looks like:



Git Concepts

To view the difference between this and the last commit, Pam uses the command `$ git diff HEAD^ -- filea`

```
diff --git a/filea b/filea
index 26a8ff9..5b06bb4 100644
--- a/filea
+++ b/filea
@@ -4,5 +4,5 @@ places = {'Mexico':'Spanish', 'United States':'English',
 for key in places:
        print key, places[key]

-for i in [4,5,6]:
+for i in [1,2,3,4,5,6]:
        print i
```

Git Concepts

Because she has configured Git to use a difftool, she also uses vimdiff with the command `$ git difftool HEAD^ -- filea` for a side-by-side comparison.

Framework For Understanding Git

Understanding Scope

- Know the difference between
  - git directory (i.e. Gitland)
  - working directory (current, local state of files)
  - The location of `HEAD` in your git directory and any local file modifications determine the state of your working directory

Understanding the Commands

Commands fall under four categories:

1. Update your **working directory** to reflect a **git directory**

```
$ git checkout master
```

2. Update a **git directory** with another **git directory**

```
$ git push origin master
```

3. Update a **git directory** with your current **working directory**

```
$ git commit
```

4. Update within a **git directory**

```
$ git merge dev
```

Next Steps

We could not cover everything, here's how to proceed:

- Understanding how Git records file states or snapshots
- Creating and using git branches
- Customizing git
- Viewing differences across file versions (i.e. diffing)
- Reverting changes

Comprehensive Resources

Many resources are available for git. Stackoverflow will answer most questions. This post is a great resource for beginners and advanced users alike.

Now **learn** Git so you can **forget** about versioning and move-on with research!

Chacon, S. and J. C. Hamano (2009). *Pro Git*, Volume 288. Springer.