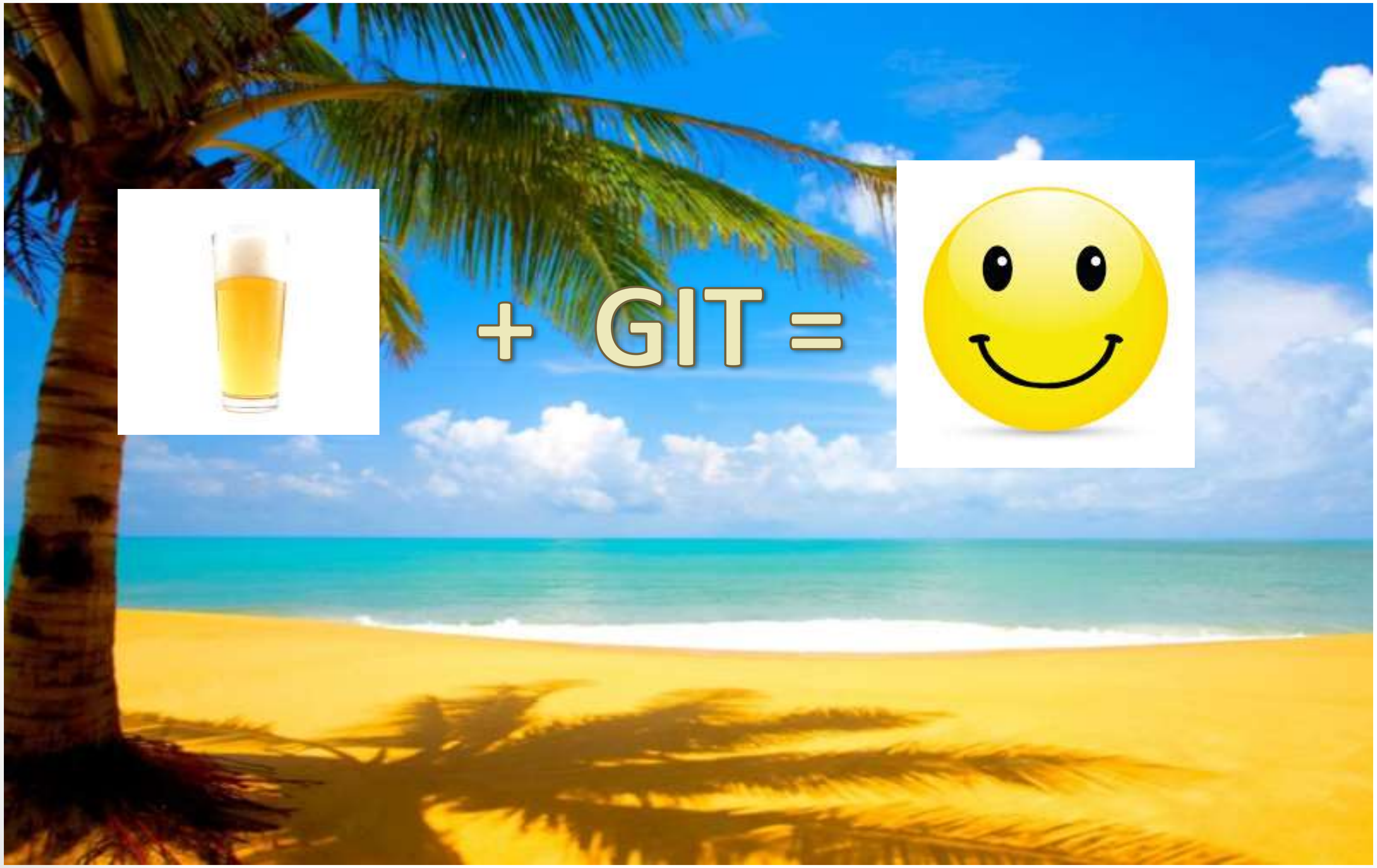
GIT

By:Naim Latifi

Hope you had a great summer !



Agenda

- VCS (Version control systems)
- What is GIT ?
- GIT branching
- GIT most common commands
- An example scenario

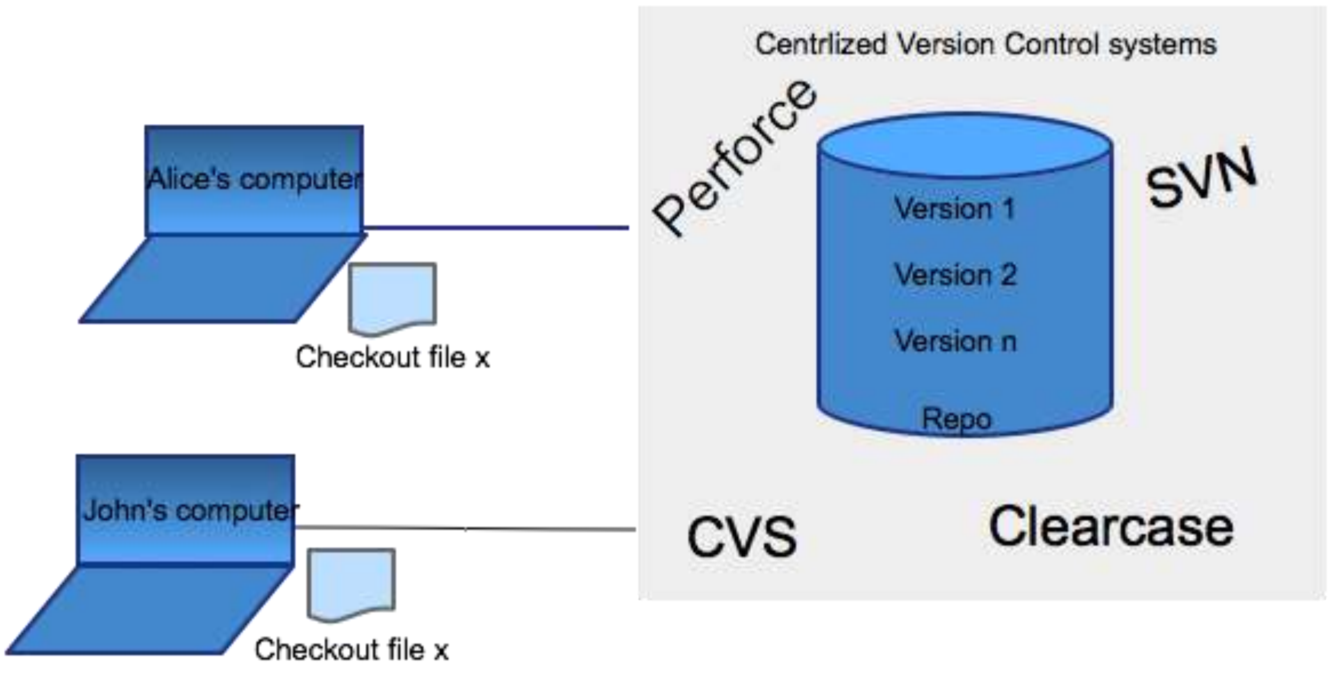
Version control systems (VCS)

- Easily management collaboration on a project by avoiding frustration of swapping files.
- Ability to have unlimited number of developers working on the same code base.
- Easily revert back your files if something happened.

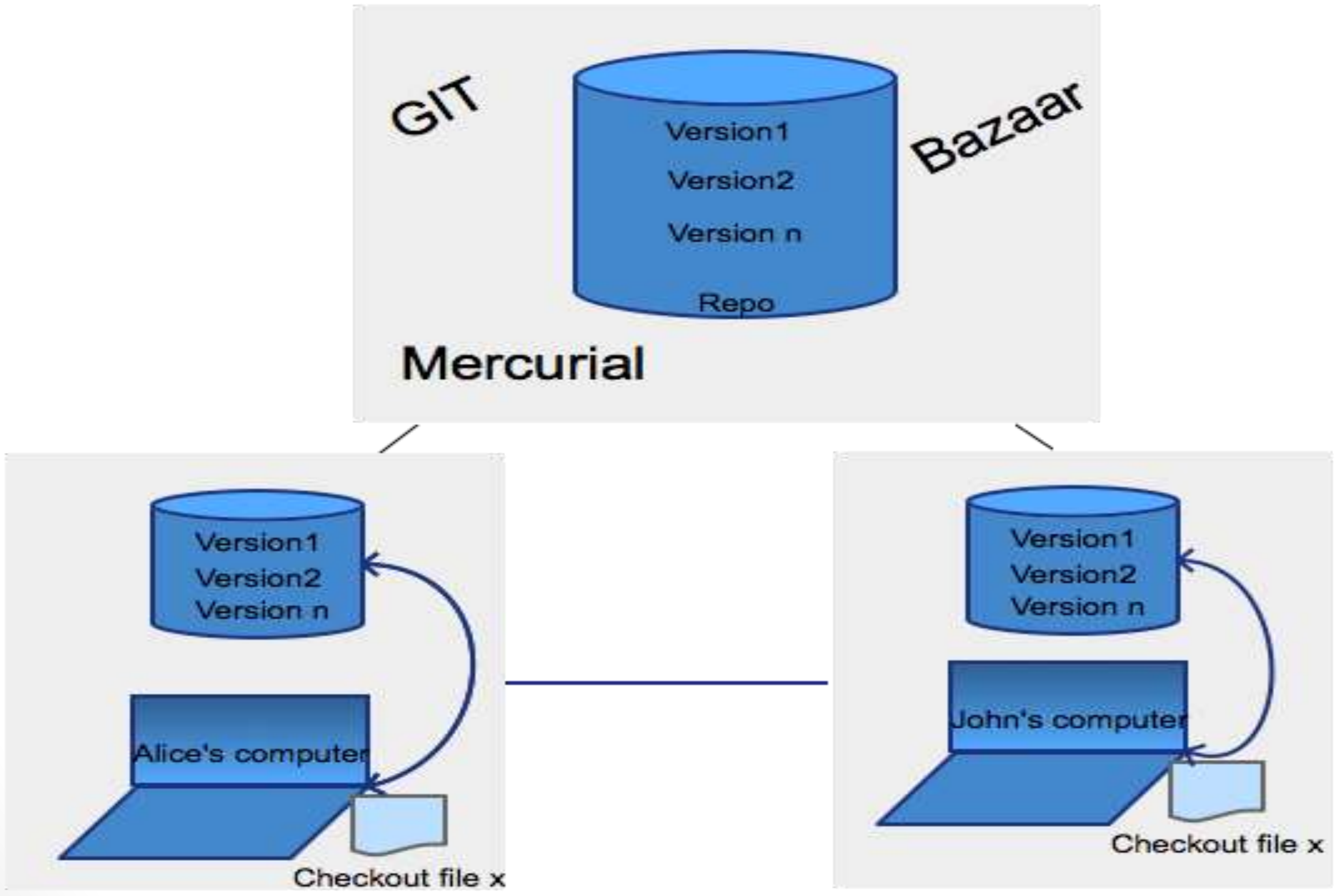
VCS (cont’d)

- **There are two main categories of version control systems**
  - Centralized version control systems
  - Distributed version control systems

Centralized VCS



Distributed VCS



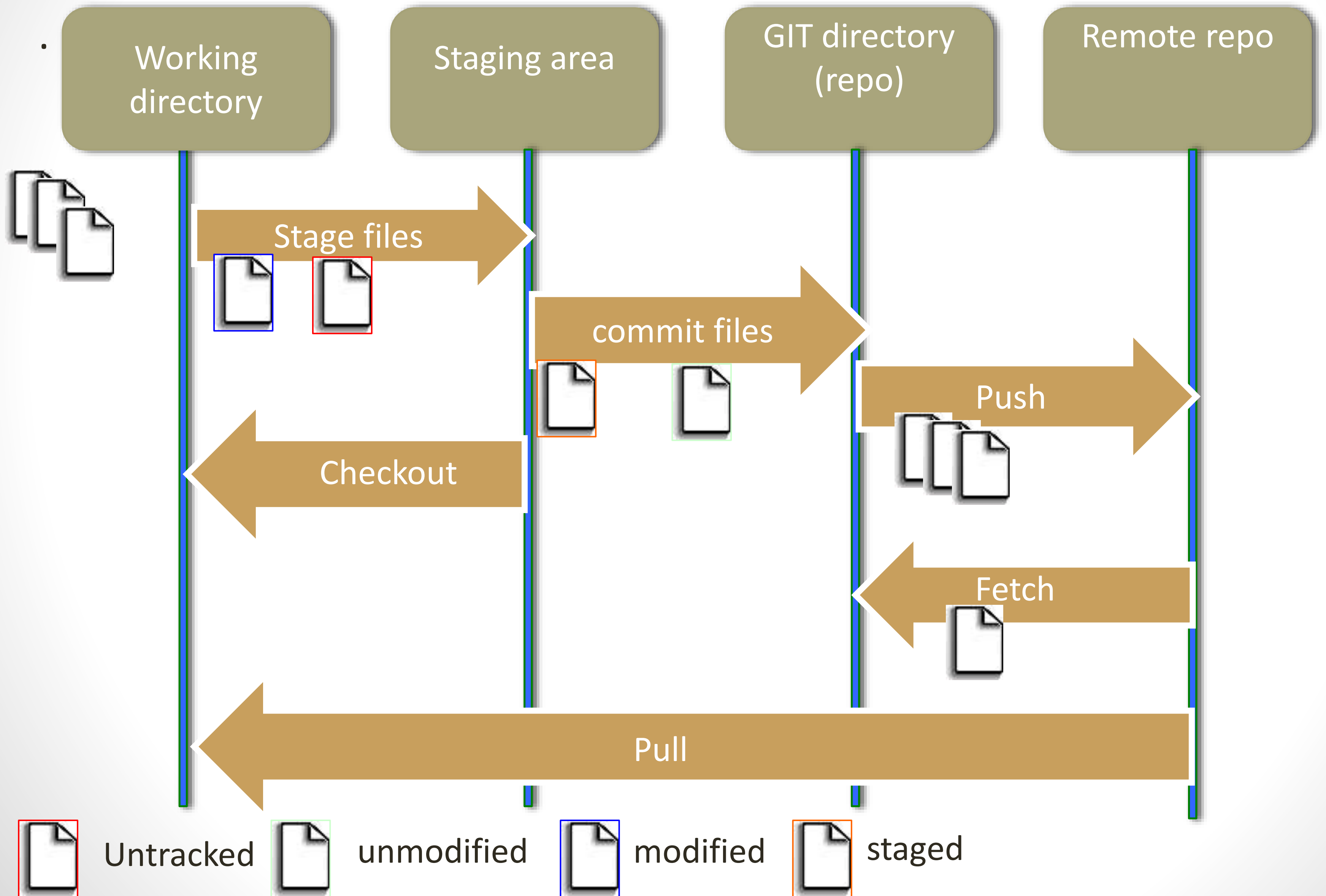
What is GIT ?

- GIT is free and open source distributed system with the emphasis on speed and data integrity.
- No centralized connectivity is needed.
- Powerful and cheap branching with easy to merge.
- Loosing work in your project is very very hard.

.git directory structure

- ├─.git
- ├─ HEAD/ (A pointer to your current branch)
- ├─ config/ (contains all configuration preferences)
- ├─ description/(description of your project)
- ├─ Index/ (is used as staging area between working directory and repo)
- ├─logs/ (keeps records to changes that are made in ref)
- ├─objects/ (all data are stored here: commits, trees and tags)
- ├─ hooks/ (shell scrips that are invoked after executing a command)
- ├─refs/ (holds your local branch remote branch and tags)

GIT state operation

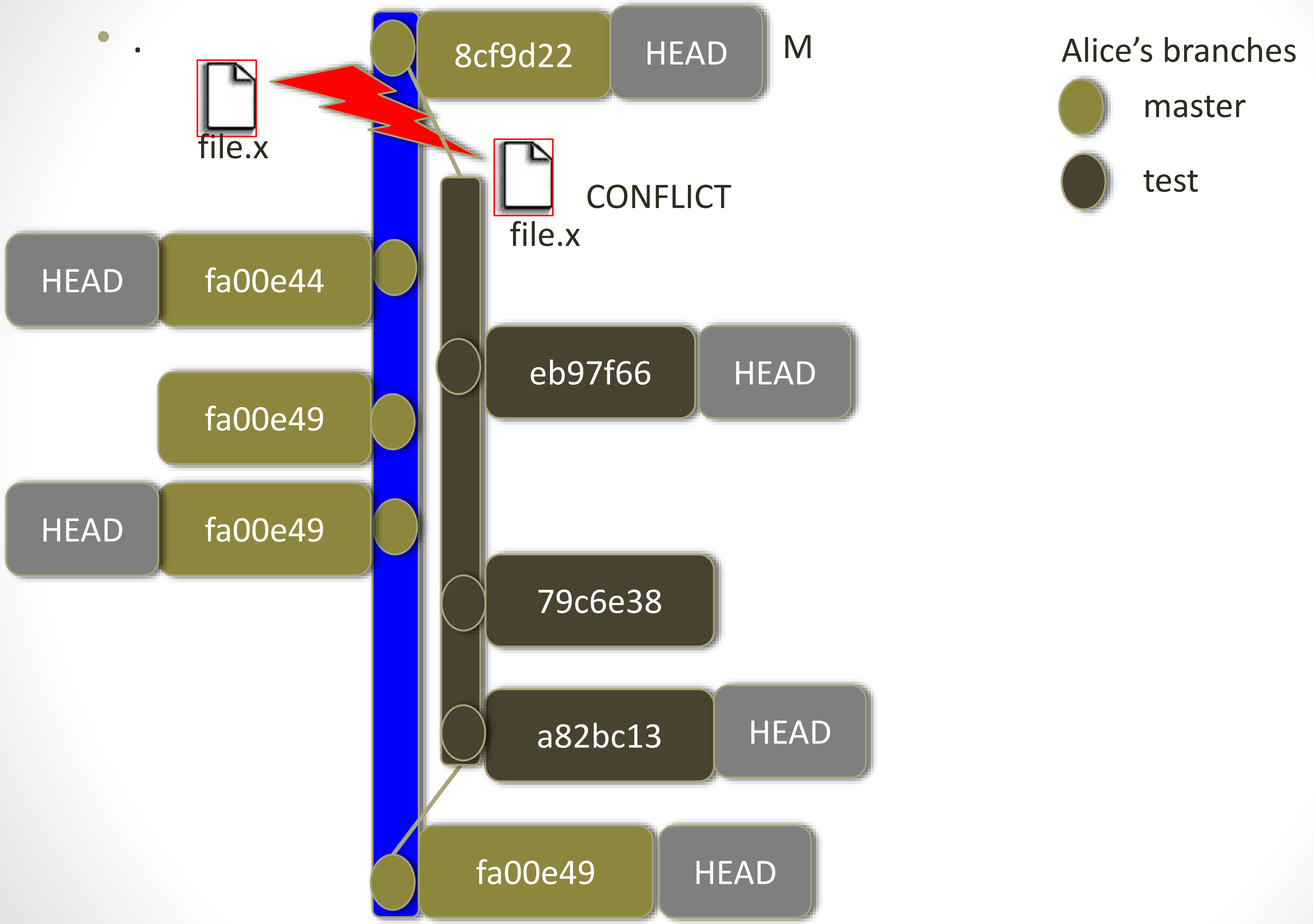


GIT branching

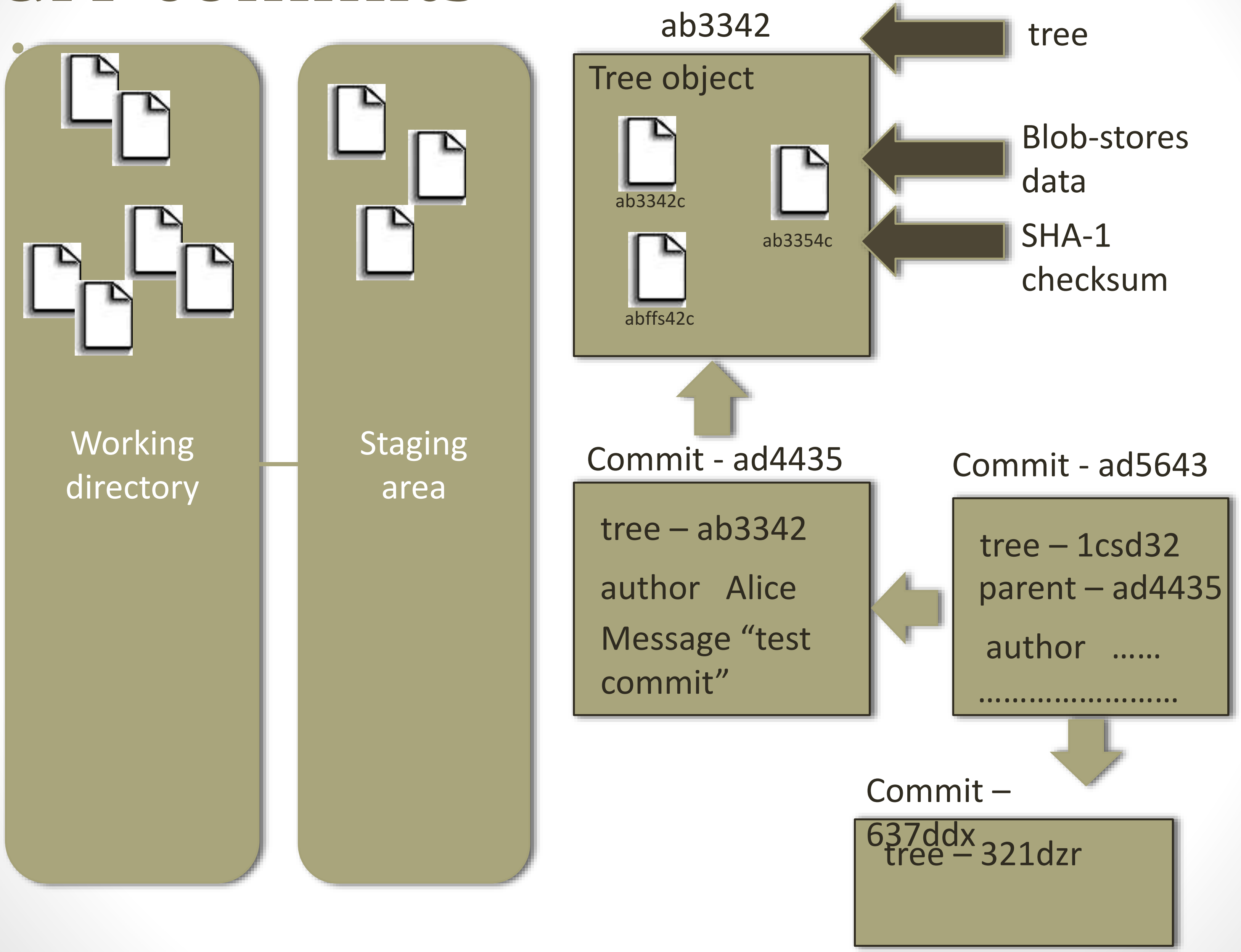
- Lightweight movable pointers for different commits.
- By default the branch name is called “master”.
- You can create as many branches as you need, merge them or delete (branches are CHEAP on GIT).
- Merging of branches is easy.

Branch & merge - a case scenario !

- .



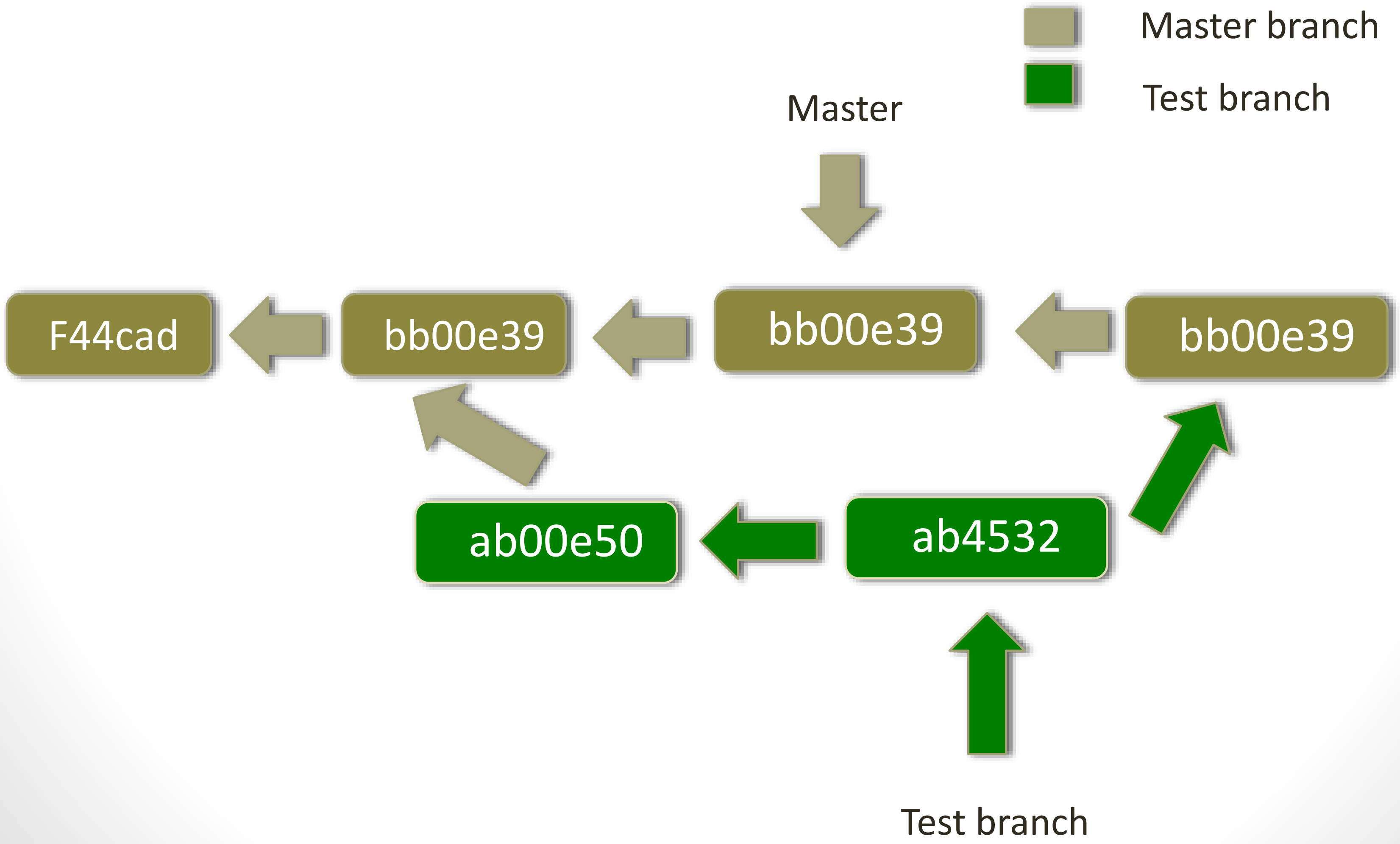
GIT commits



ab3342
tree
Tree object
Blob-stores data
ab3342c
ab3354c
SHA-1 checksum
abffs42c
Working directory
Staging area
Commit - ad4435
tree – ab3342
author Alice
Message “test commit”
Commit - ad5643
tree – 1csd32
parent – ad4435
author ……
……………………
Commit – 637ddx
tree – 321dzr

GIT three way merge

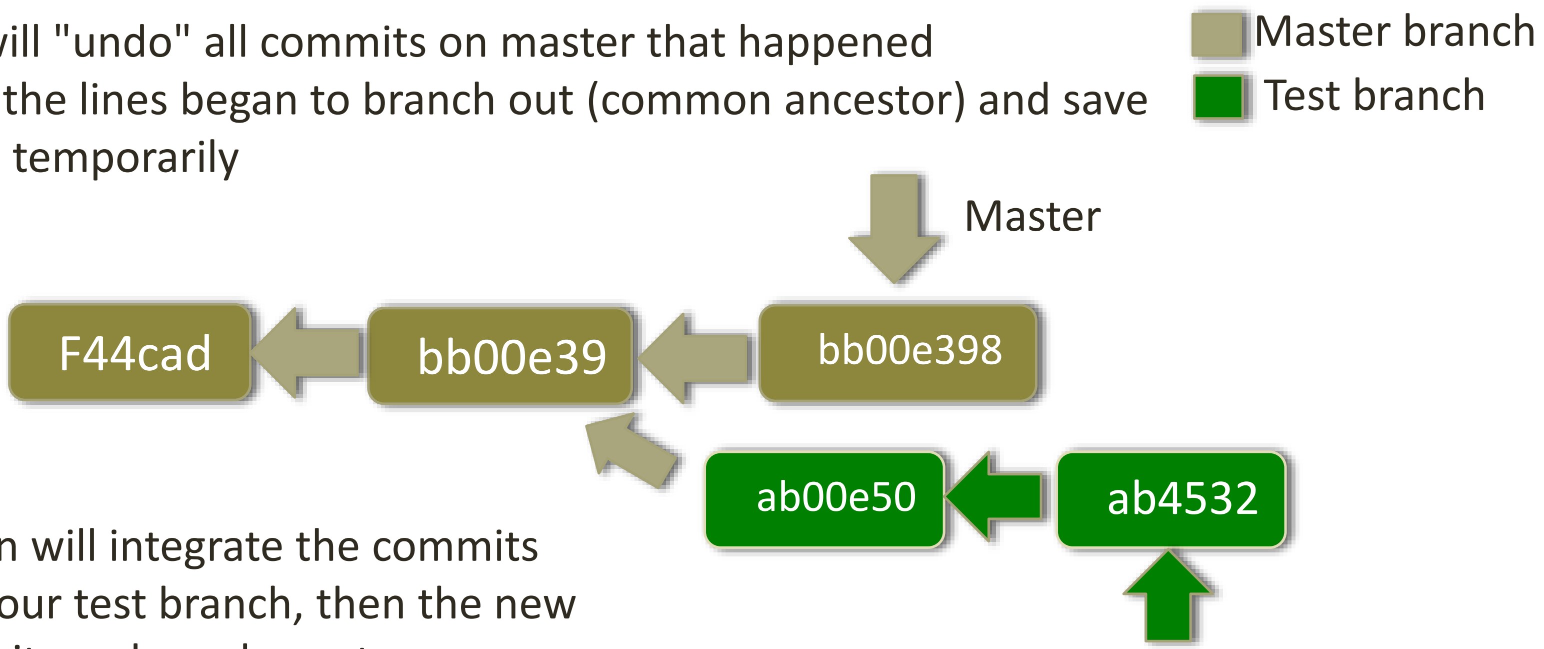
It takes the two latest commits from branch (bb00e39, ab4532) and the most common ancestor of the two (bb00e39) and creates a new commit (bb00e39) after successful merge



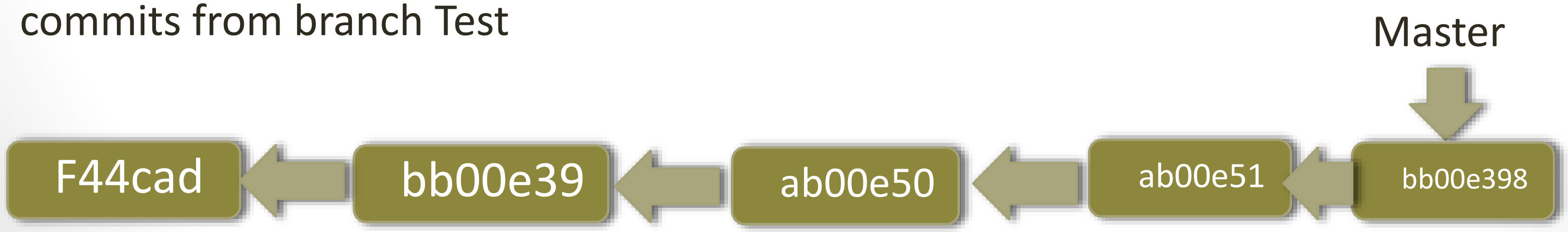
GIT rebasing

The major benefit of rebasing is that you get a much cleaner project history

GIT will "undo" all commits on master that happened after the lines began to branch out (common ancestor) and save them temporarily



It then will integrate the commits from our test branch, then the new commits on branch master are reapplied on top of integrated commits from branch Test



GIT most common commands

- INIT
- CLONE
- STATUS
- ADD
- CHECKOUT
- COMMIT
- LOG
- REMOTE
- FETCH
- PUSH
- PULL

(cont'd)

```
$ git init
```

creates a local repository

```
$ git clone path_repi
```

Checkout a repository and create a local copy of it.

(cont’d)

```
$ git status
```

To see the state of your files (modified , added , remote..)

```
$ git add ”file_name”
```

add files to your staging area

(cont'd)

$ git checkout

Checkout a tracked file to untracked file. This command is also used for switching between branches

$ git commit –m ”message”

Commit a stage file

(cont’d)

```
$ git log
```

To get the history of your commits or commits made by someone else

```
$ git remote ’origin’
```

Shows the remote repository name that by default is origin

(cont'd)

```
$ git fetch "remote_name"
```

Updates remote repository without merging with your local

```
$ git push "remote_name
```

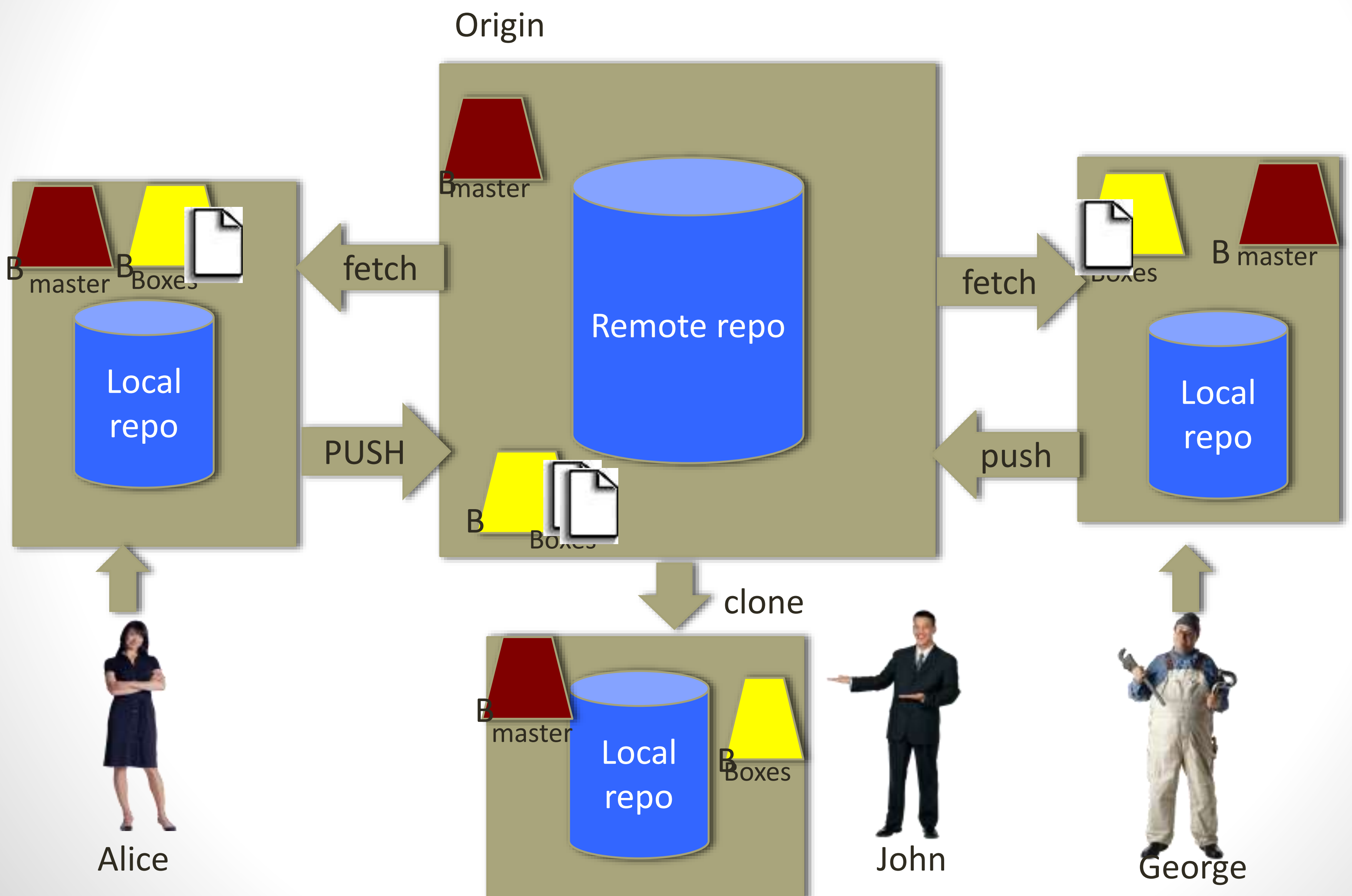
pushes your commits, branches etc. to remote repository

(cont'd)

$ git pull "remote_name"

Updates remote repository and merge changes with your local repo

An example scenario



Thank you for your time !

QUESTIONS