Softwareontwikkeling 4

Git

Tim Dams

Overview

- Introduction: what is version control
- GIT
- Getting started
- References:
  - Git commands
  - Exercise

Introduction

VERSION CONTROL

Why Source Control?

- How do two people collaborate on the same code?
  - Email?
  - Dropbox?

Why Source Control?

- How do two people collaborate on the same code?
  - Email?
  - Dropbox?
- How do two hundred people collaborate?

Why Source Control?

- How do we backup our code?
- How do we get a history of changes?

Why Source Control?

- How do we make big or risky code changes without affecting the *stable* version?
- How do we work on new versions and still support old versions?

What is Version/Source Control?

- Manages file sharing for **Concurrent Development**

- Keeps track of changes with **Version Control**

Popular version control systems

- SubVersion (SVN)
- Git
- Mercurial (Hg)
- CVS
- Bazaar
- etc.

Concurrent Development

- Server holds all original files of a project
  Gives out copies to participants (clients)
- Participants modify their copies
  Submit their changes to server
- Automatically merges changes into original files. Huge!
- Conflicts only occur when modifications are done
  - by more then one participant
  - at the same location in their respective copies.
  - Then participants have to manually resolve such conflicts. Rare!
- Powerful edit and merge tools help make this task easy

Version Control

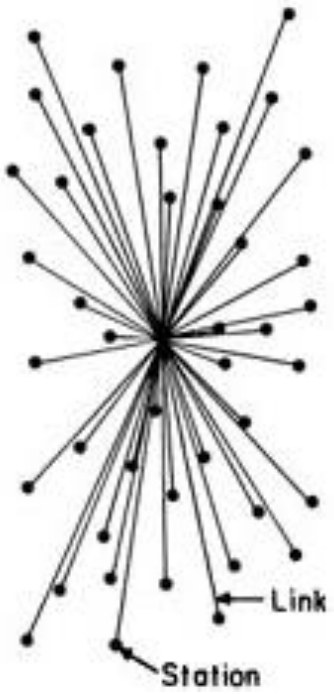
- SVN/Git keeps log of any changes made to any file. Ever!
  Also keeps copies of those changes. For ever!

- Participants can go back and receive older versions of a file
  or even an older version of an entire project state

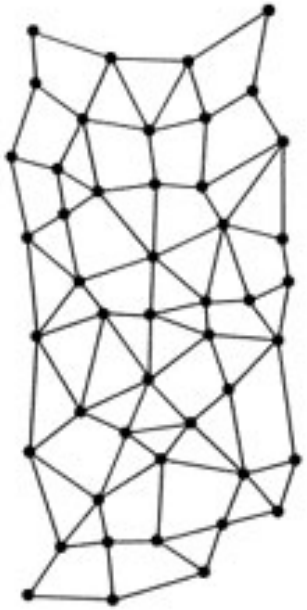
- The current version number of your project in SVN is #5
  In the future you can always load the project exactly as it is today by requesting project version #5; et voila you can run an age old demo!

Distributed or centralized?

Most Version Control Systems



Linus' Vision of Git



Reality of Distributed Systems

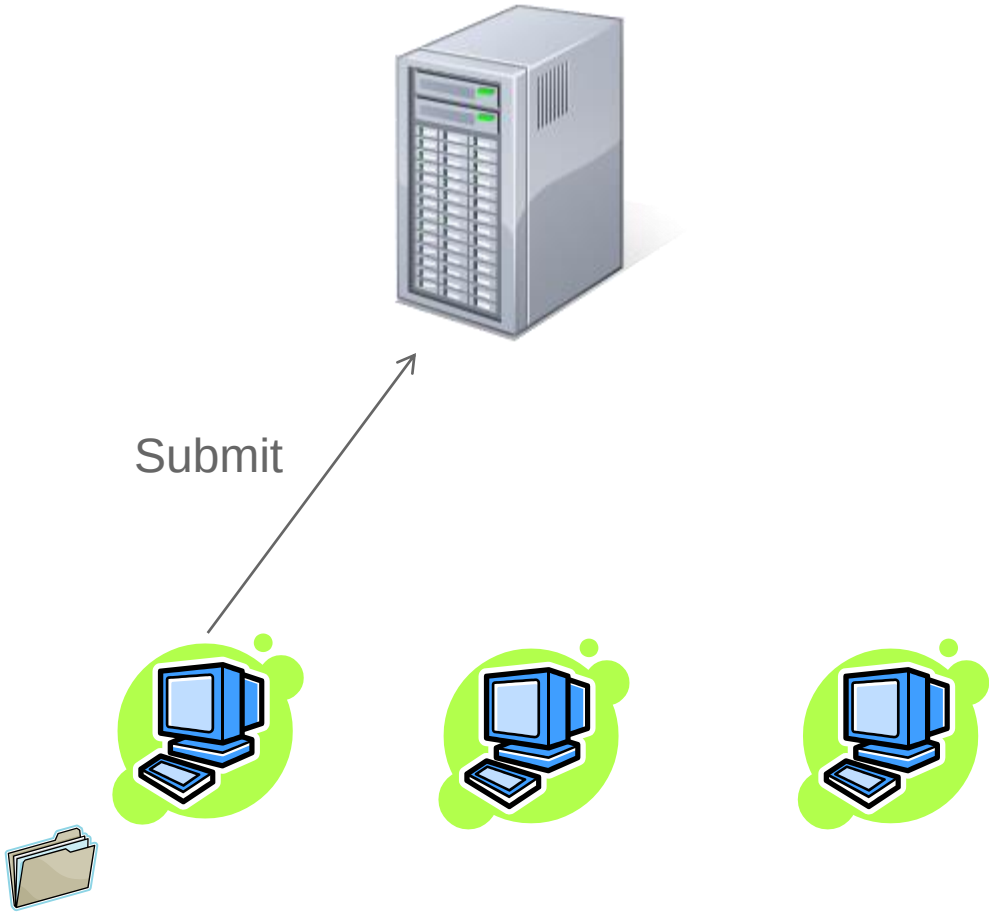
Centralized

Server holds a centralized *repository*

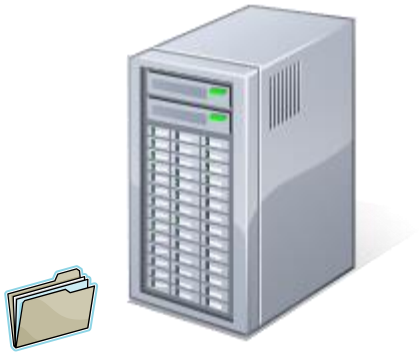
Clients have a revision

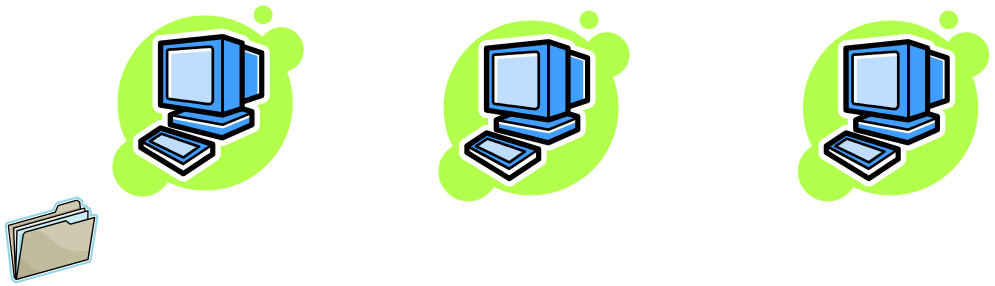
Centralized

Centralized

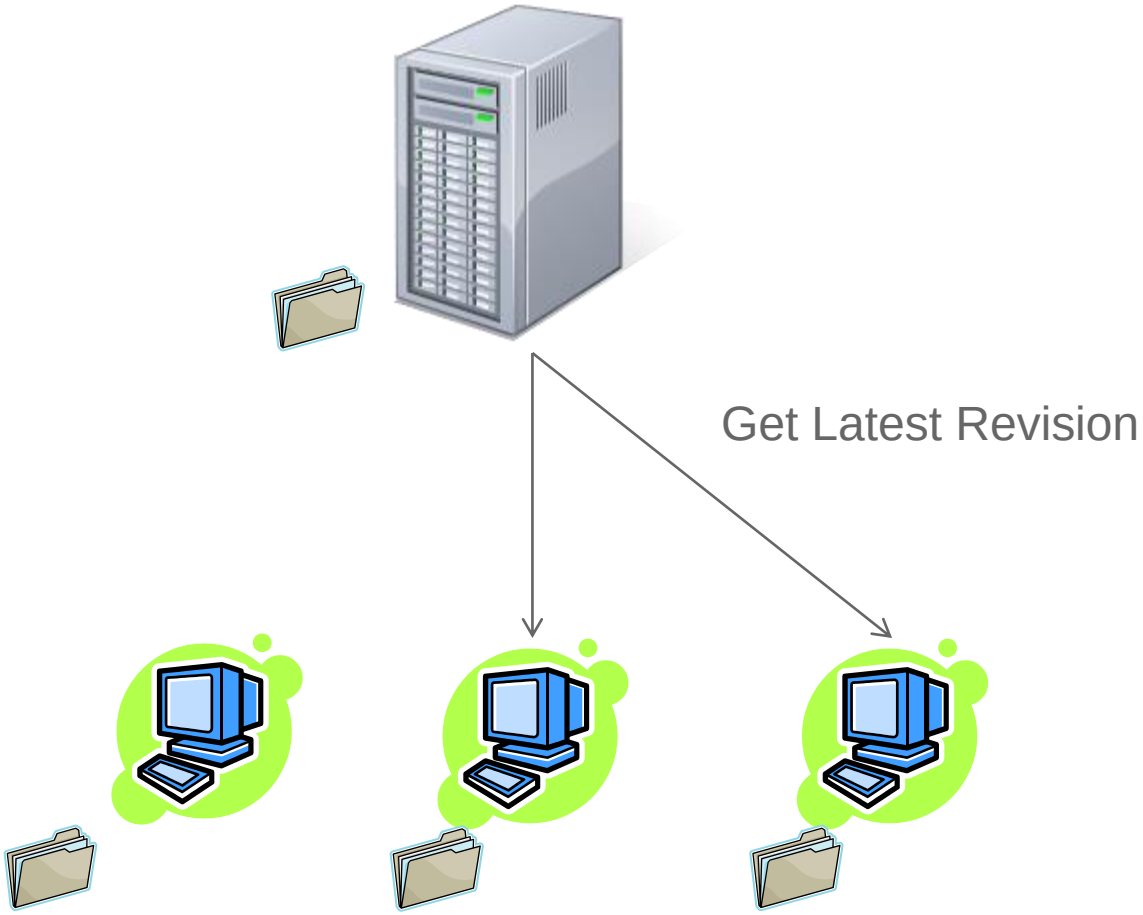


Centralized

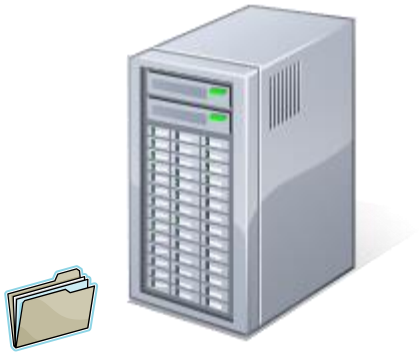




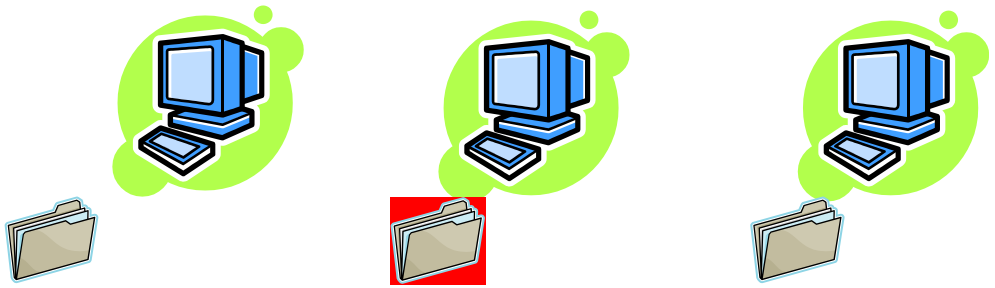
Centralized



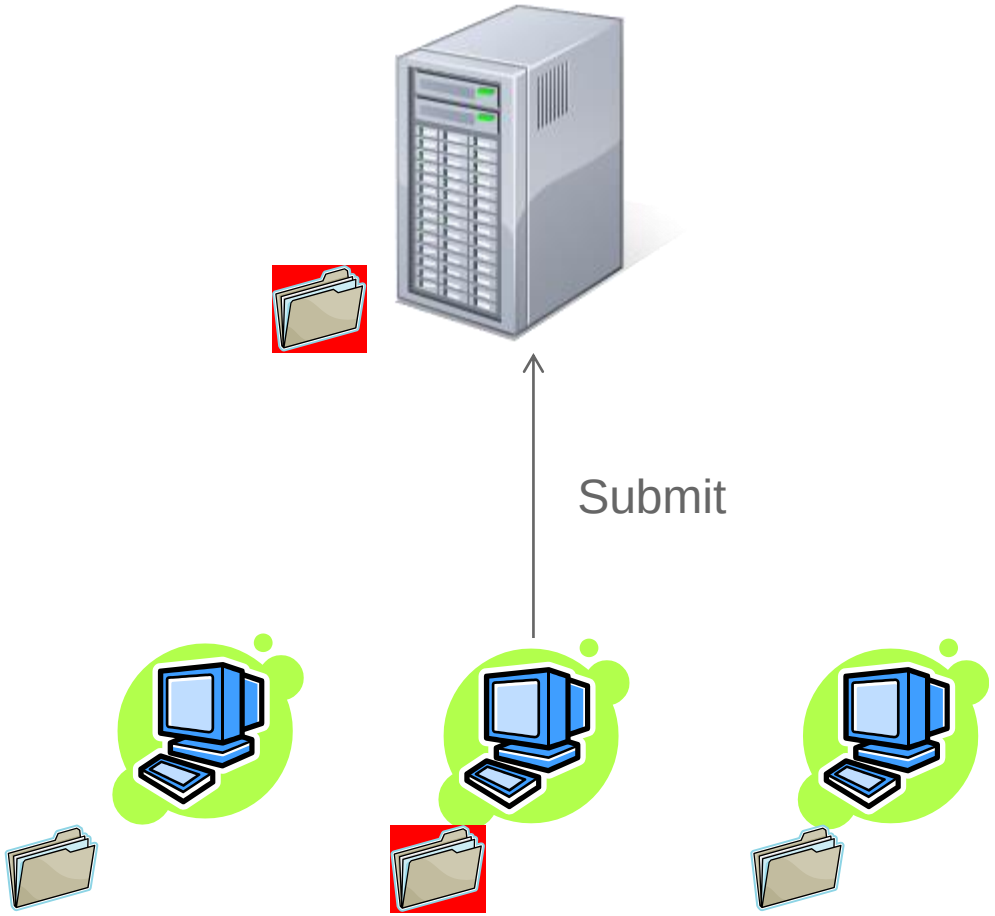
Centralized



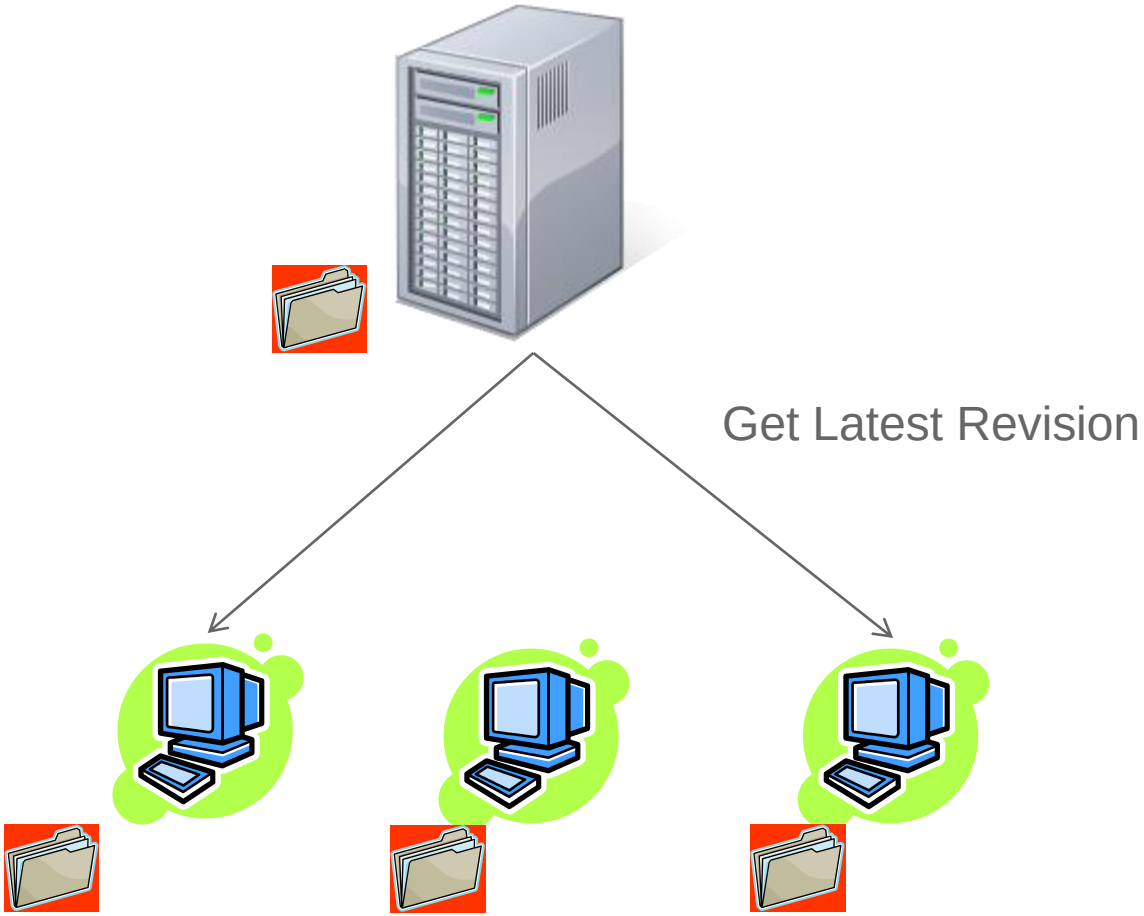
Modify/add/remove files



Centralized



Centralized



GIT

What is git?

- Git was created by Linus Torvalds around 2005 and is based on ‘bitkeeper’.
- Git is a **distributed version control** system
- Comparable to CVS, SVN, PVCS, VSS but also wholly and entirely different.
- Open source.

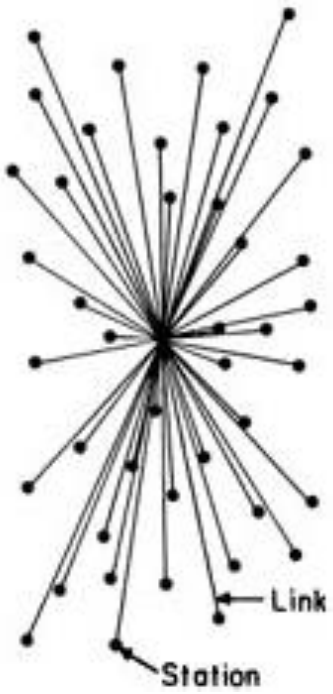
The End

When I say I hate CVS with a passion, I have to also say that if there are any SVN [Subversion] users in the audience, you might want to leave. Because my hatred of CVS has meant that I see Subversion as being the most pointless project ever started. The slogan of Subversion for a while was "CVS done right", or something like that, and if you start with that kind of slogan, there's nowhere you can go. There is no way to do CVS right.

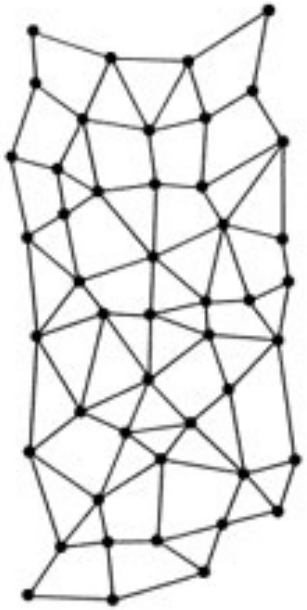
--Linus Torvalds, as quoted in Wikipedia

Distributed?

Most Version Control Systems

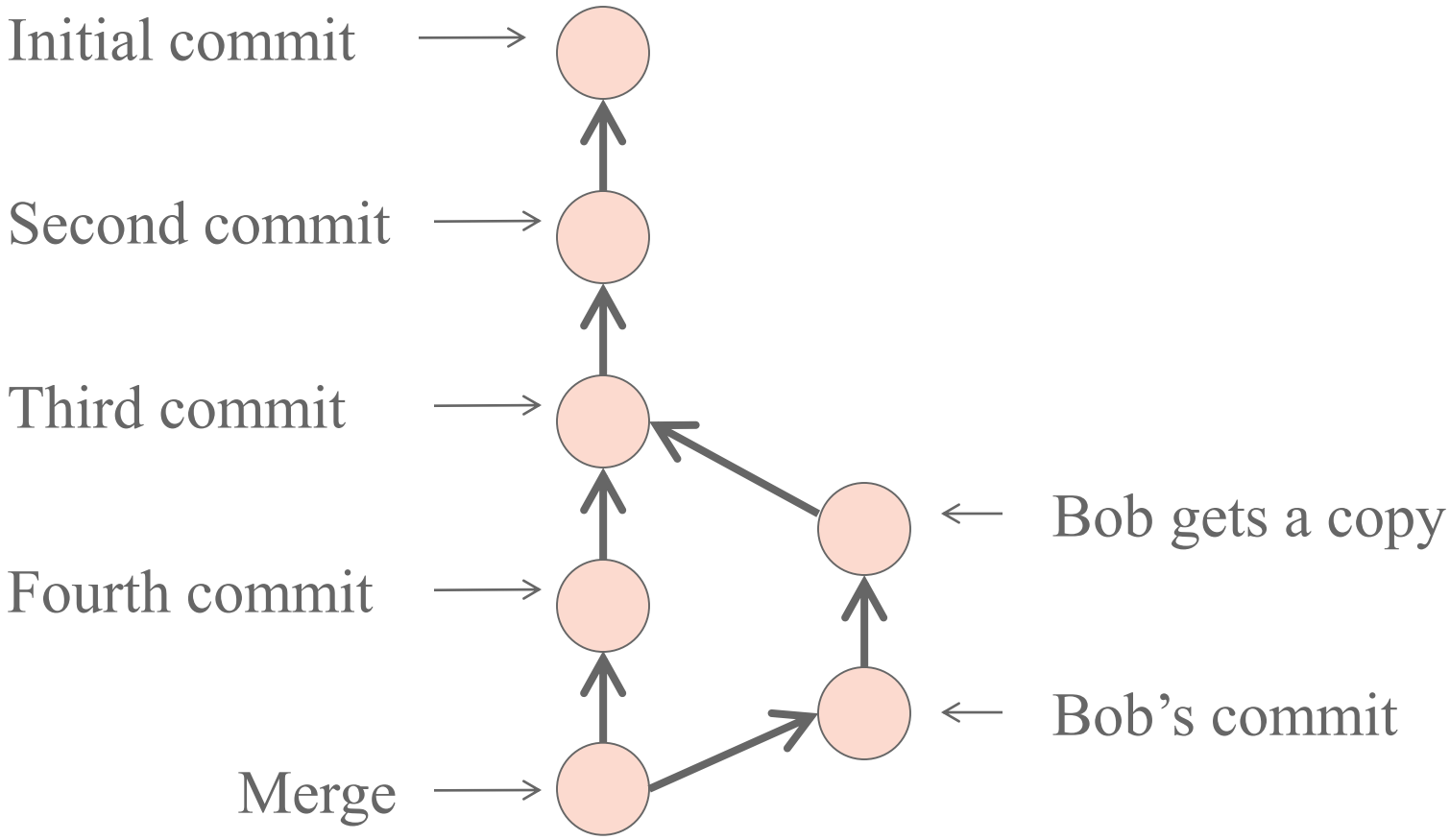


Linus' Vision of Git



Reality of Distributed Systems

Multiple versions



Distributed

All machines have a full copy of the repository

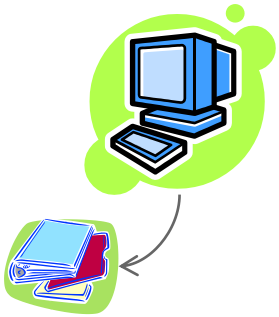
Repositories can be *cloned*

Repositories can be *pushed* to and *pulled* from other machines

Distributed

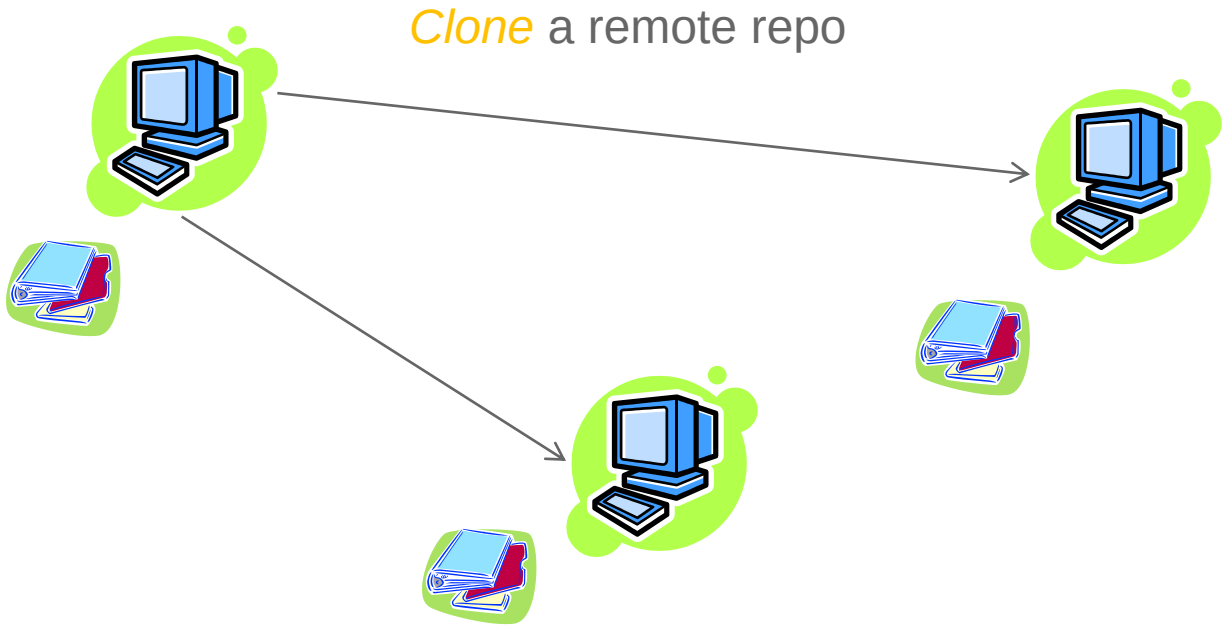
Create a local repo
Locally *commit* changes

```
git init
// Add files to directory
git add *
git commit -a -m '<description>'
```



Distributed

```
git clone git@github.com:CIS565-Spring-2012/cis565testHomework.git
```



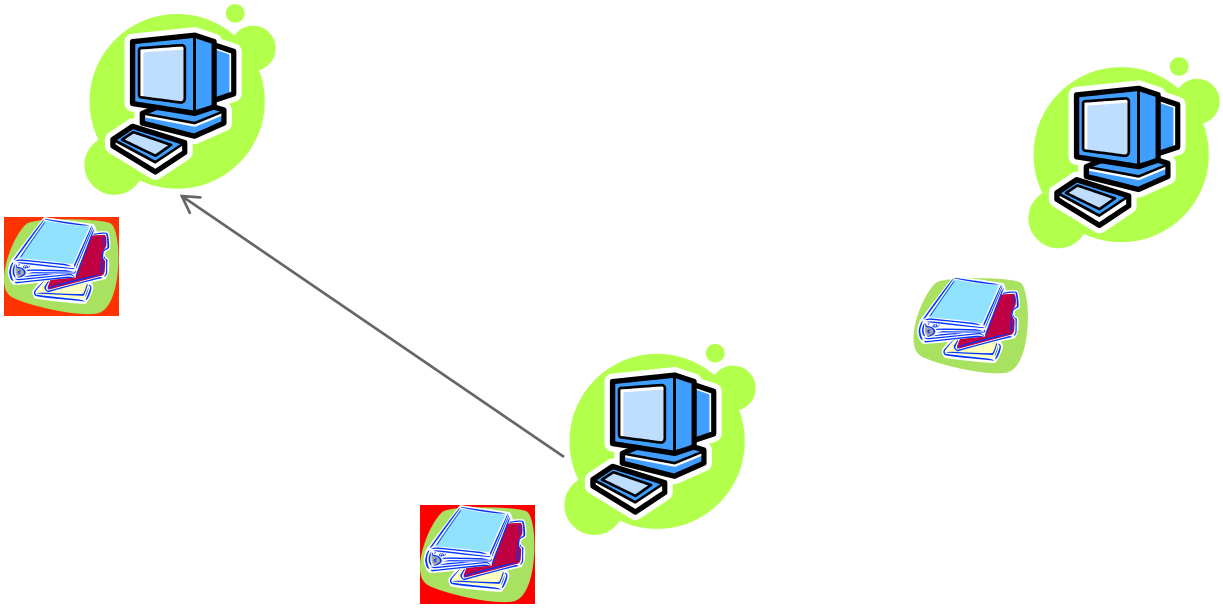
Distributed

```
git commit -a -m '<description>'
```

Locally *commit* changes

Distributed

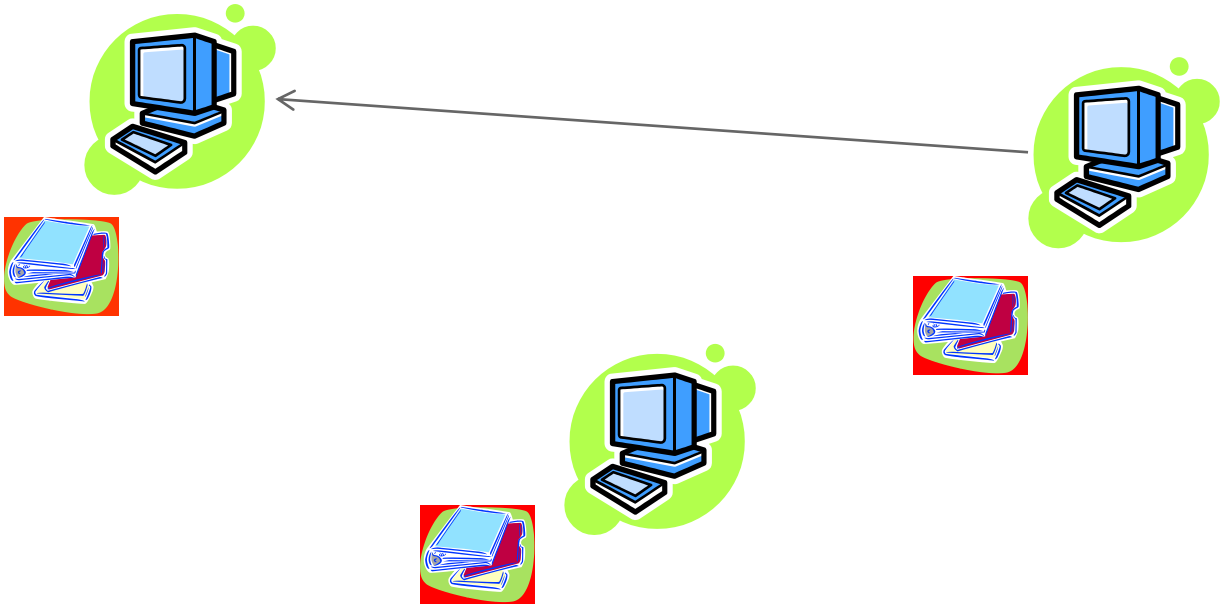
```
git push
```



Push changes to other repos

Distributed

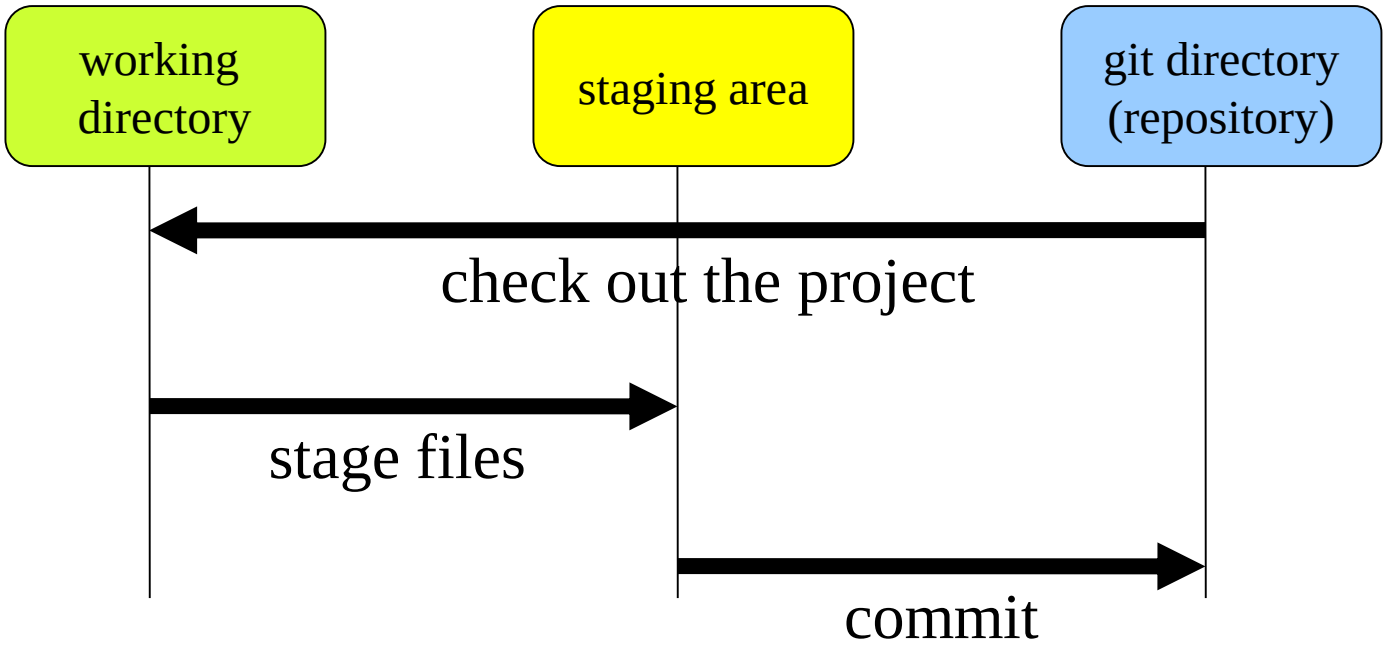
```
git pull
```



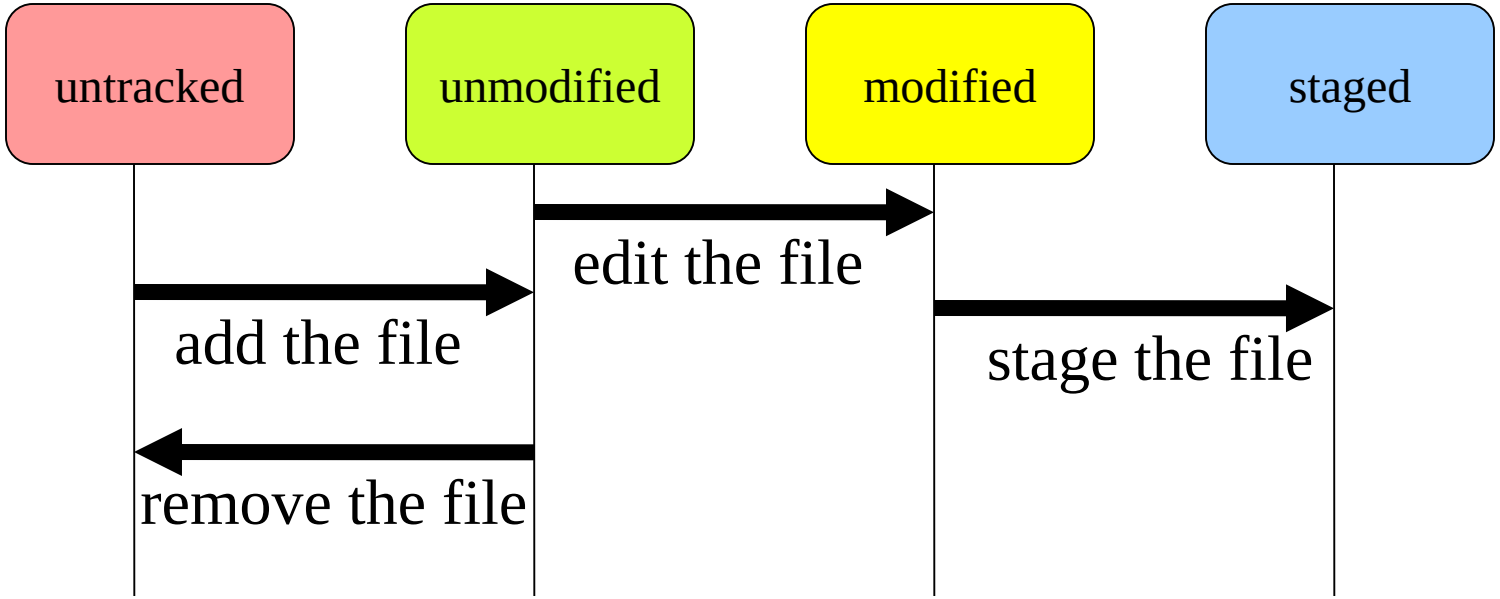
Pull changes from other repos

3 States of a File in Git

- **Modified • Staged • Committed**



File Status Lifecycle



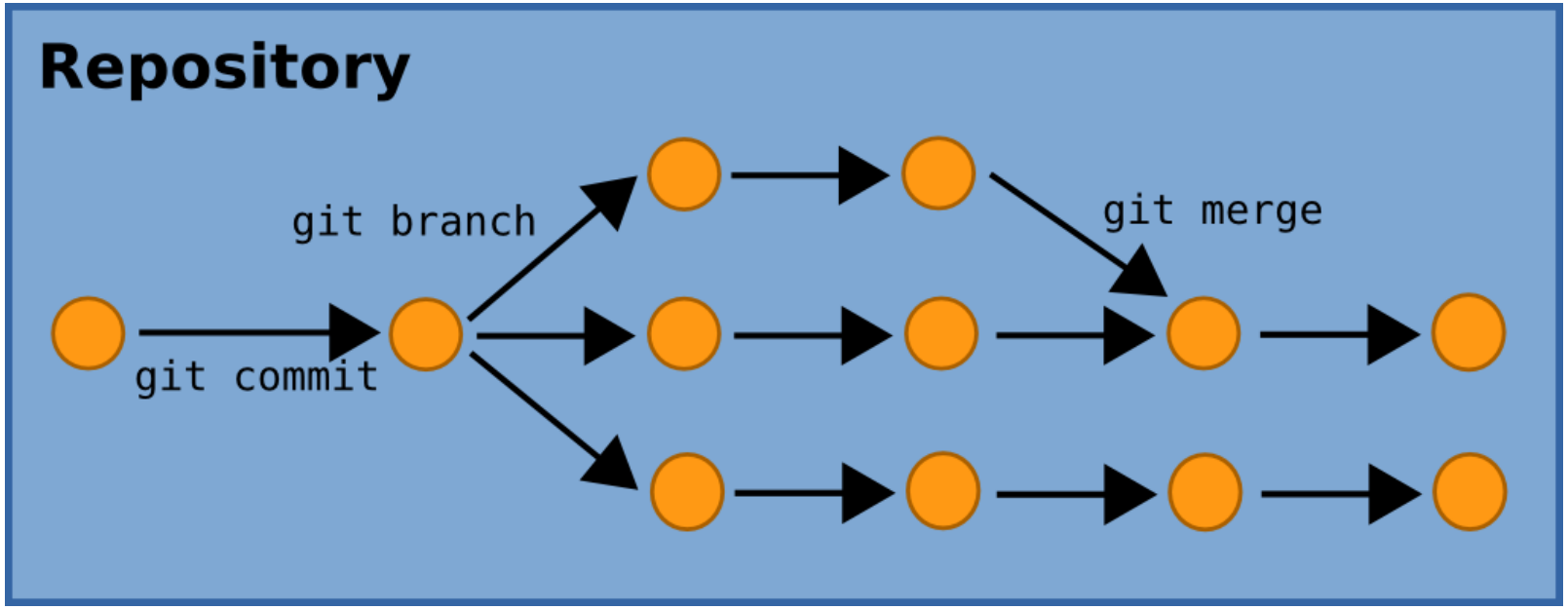
Some Commands

- Getting a Repository
  - git init
  - git clone
- Commits
  - git add
  - git commit
- Getting information
  - git help
  - git status
  - git diff
  - git log
  - git show

Undoing What is Done

- git checkout
  - Used to checkout a specific version/branch of the tree
- git reset
  - Moves the tree back to a certain specified version
  - Use the --force to ignore working changes
- git revert
  - Reverts a commit
  - Does not delete the commit object, just applies a patch
  - Reverts can themselves be reverted!
- Git never deletes a commit object
  - It is very hard to shoot yourself in the foot!

Branching and merging



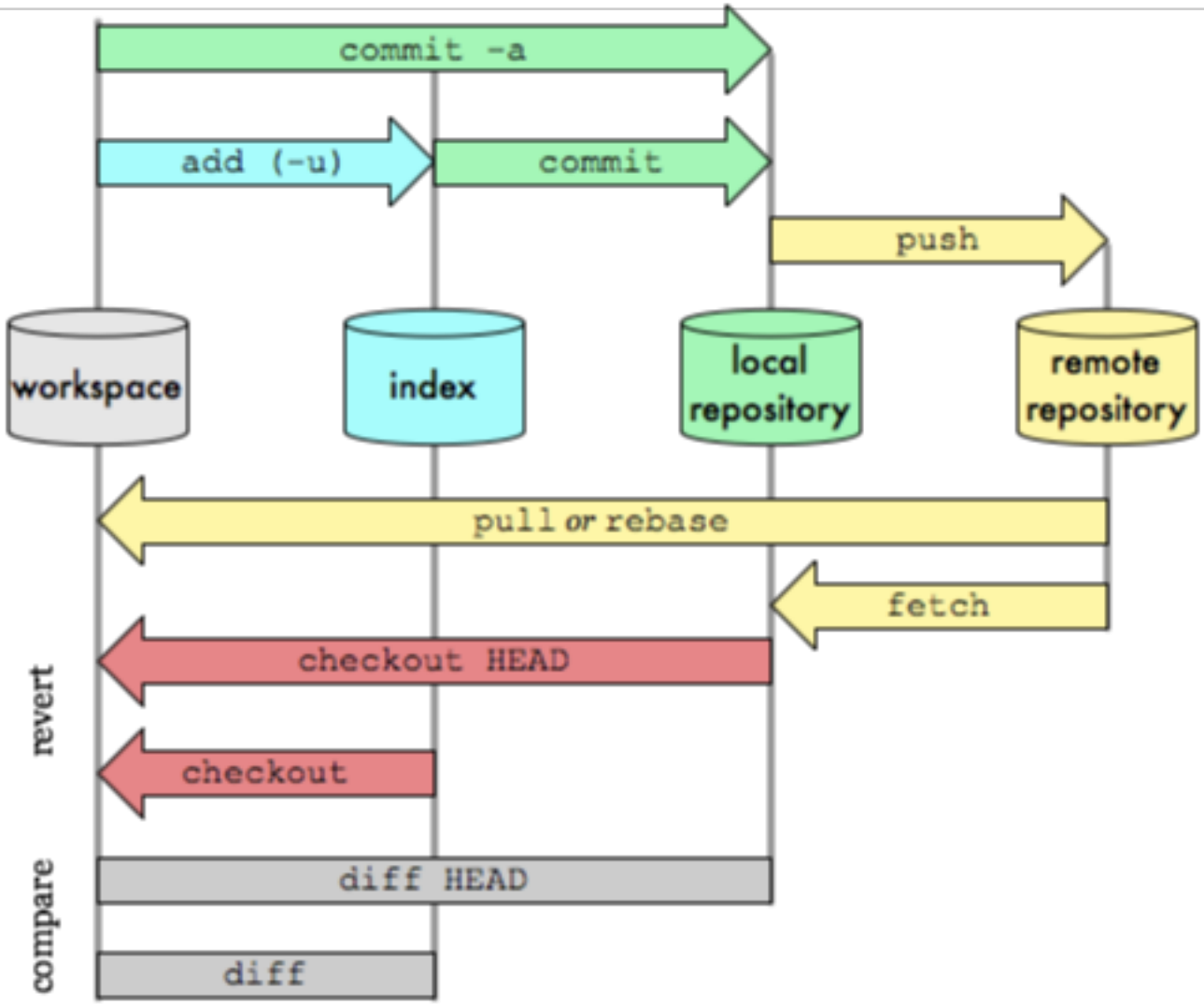
Using Remote

- Use git clone to replicate repository
- Get changes with
  - git fetch (fetches and merges)
  - git pull
- Propagate changes with
  - git push
- Protocols
  - Local filesystem
  - SSH
  - Rsync
  - HTTP
  - Git protocol

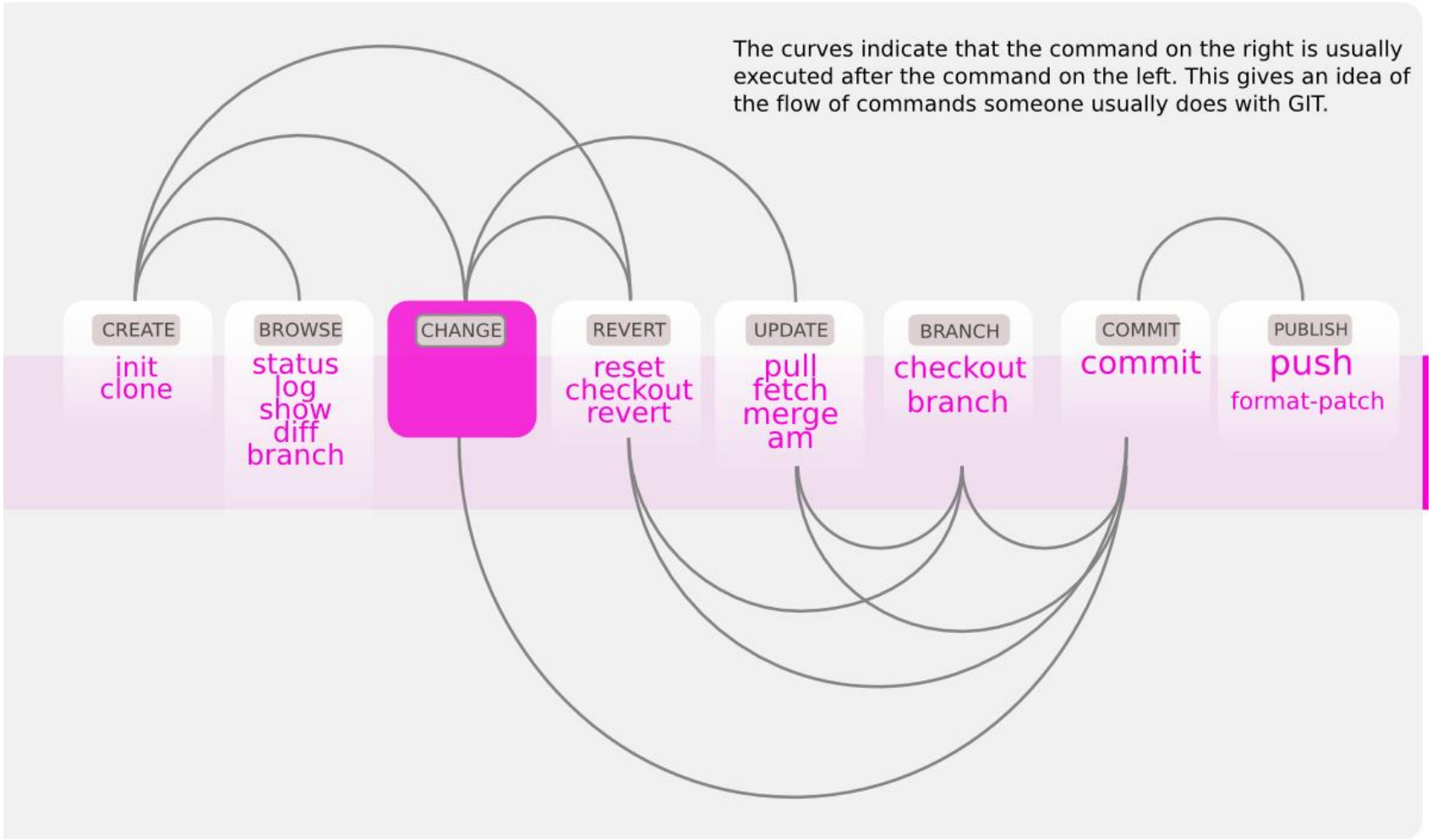
Cloning our Repository

- ***git clone*** *first-git-repo*
  - Now have a full git repository to work with
- Changes are pushed back with ***git push***
  - Pushing changes WILL NOT change working copy on the repository being worked on
- Branches can be based off of remote branches
  - *git branch --track new-branch remote/branch*
- Remote configuration information stored in *.git/config*
  - Can have multiple remote backends!

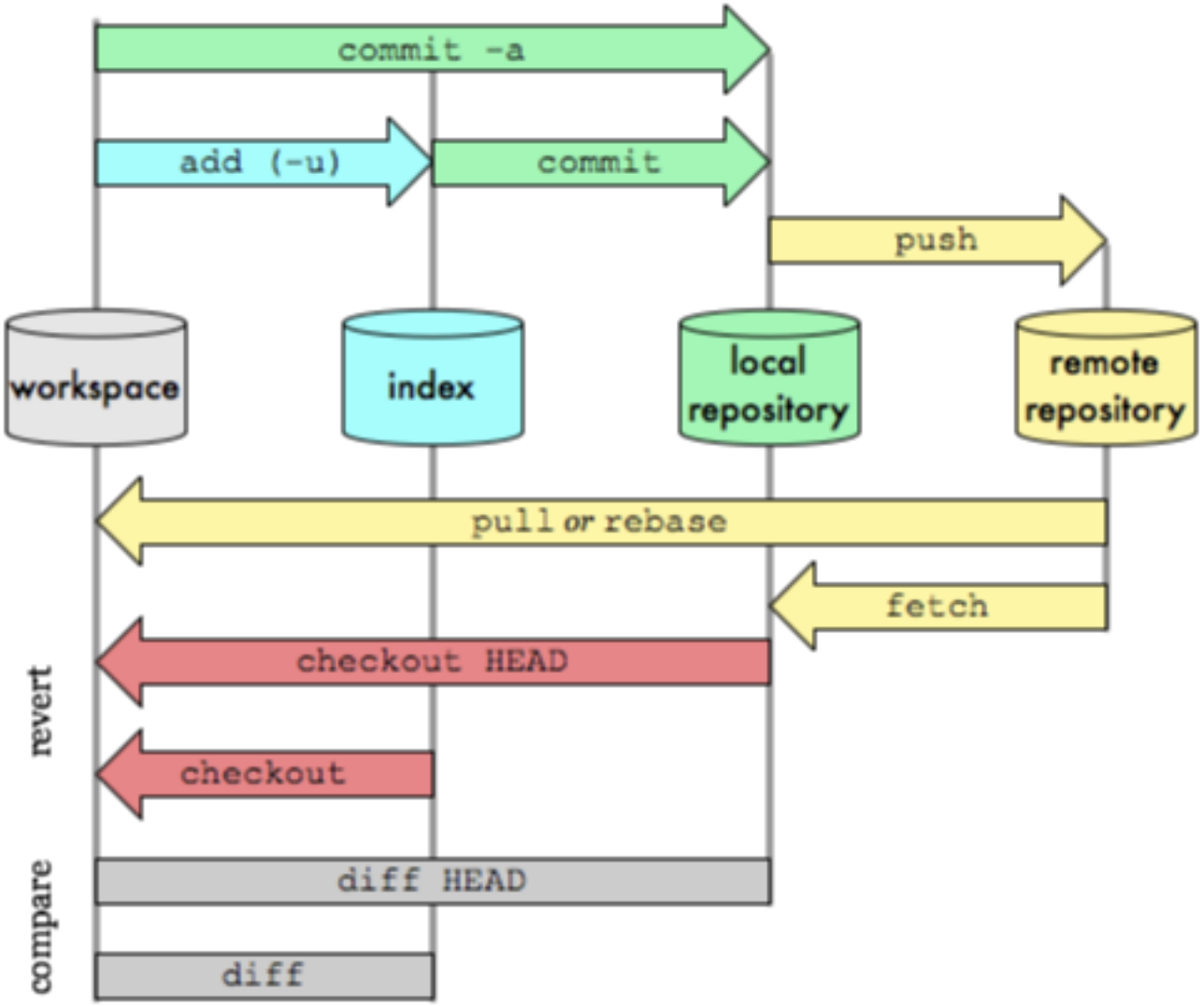
Git transport commands



Commands Sequence

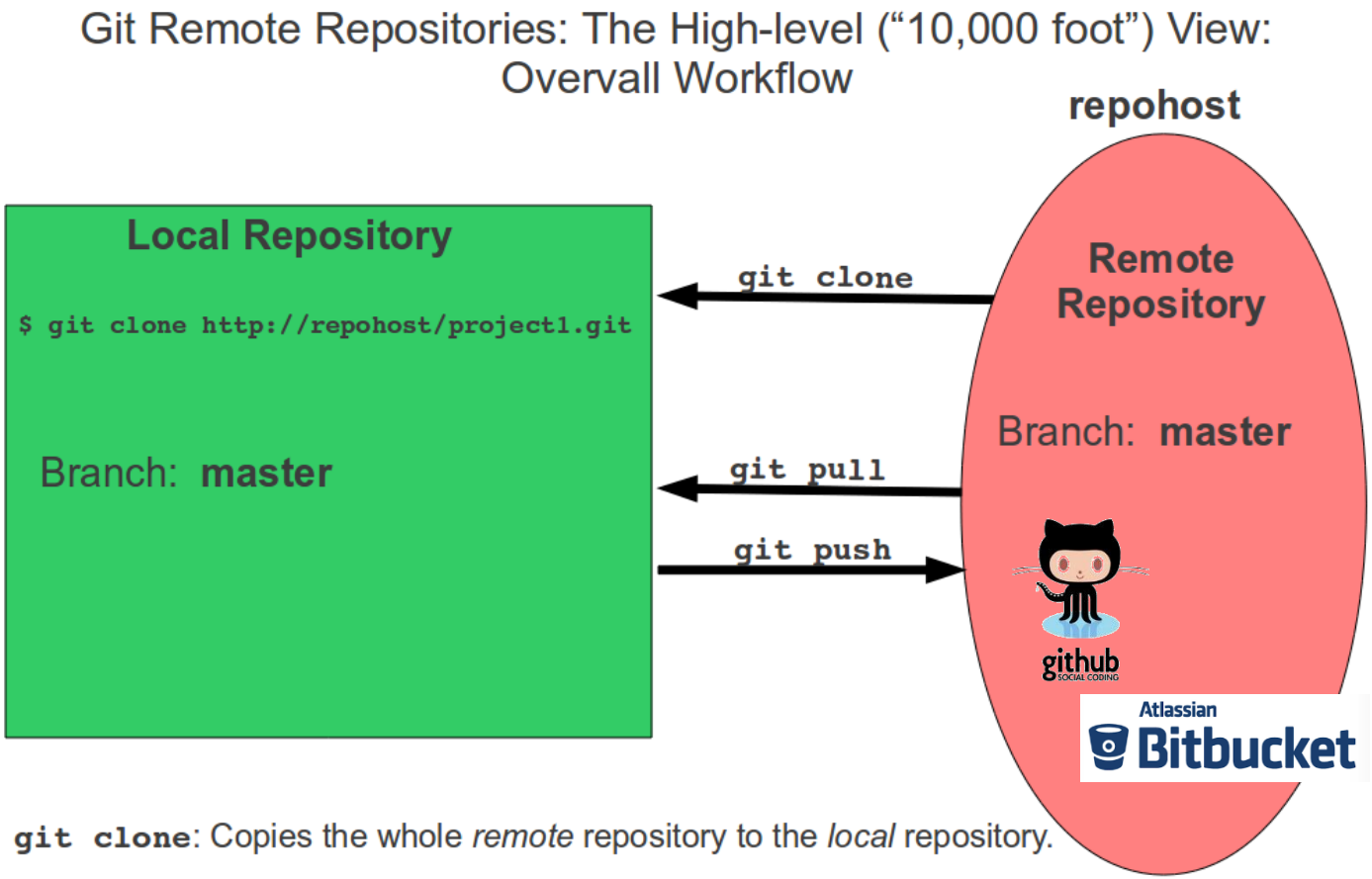


A simple Git workflow



GETTING STARTED

Overview



`git clone`: Copies the whole *remote* repository to the *local* repository.

`git pull`: Retrieves any updates from the remote repository that aren't yet in the local repository and merges them into the local repository.

`git push`: Publishes updates from the local repository to the remote repository

Getting started

- Software:
  - I recommend SmartGit + http://git-scm.com/
  - Learn yourself: http://try.github.io/levels/1/challenges/1
  - Great tutorial: http://marklodato.github.io/visual-git-guide/index-en.html
  - More early start info:
    - https://help.github.com/
    - http://rogerdudler.github.com/git-guide/
    - http://gitimmersion.com

Online git repos

- Free git-repos:
  - http://www.Github.com (projects will be publicly visible, unless paid)
  - http://www.Bitbucket.org (git)
  - http://www.projectlocker.com/ (git & svn)
  - http://www.codeplex.com (supports .NET ClickOnce)
  - http://code.google.com (git & svn)
  - http://www.sourceforge.com (get & svn)
  - http://tfs.visualstudio.com/ (git &

Resolving conflicts

- http://www.syntevo.com/smartgithg/howtos.html
- http://www.slightlymagic.net/wiki/Forge:_How_to_Install_and_Use_SmartGit

.gitignore

- Good practice to first add a .gitignore file.
  - Lists files, extensions to ignore
  - E.g. build files

- https://github.com/github/gitignore

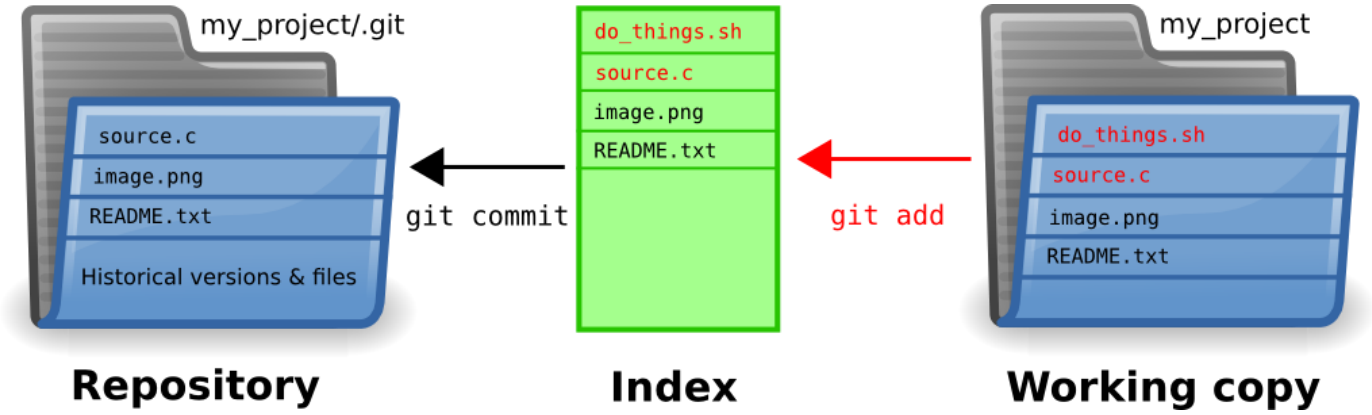
- So way to go:
  1. Create new repo (*git init*)
  2. Add correct .gitignore files (*git add .gitignore*)
  3. Commit changes (*git commit –m “Let’s start, .gitignore added”*)
  4. Start working

More info:

- Best way to learn git: USE IT!

- **From now on: ALLE Project ict4 én lab softwareontwikkelings-projecten moeten git-history hebben.**

1. Create new repo (*git init*)
2. Add correct .gitignore files (*git add .gitignore*)
3. Commit changes (*git commit –m “Let’s start, .gitignore added”*)
4. Start working



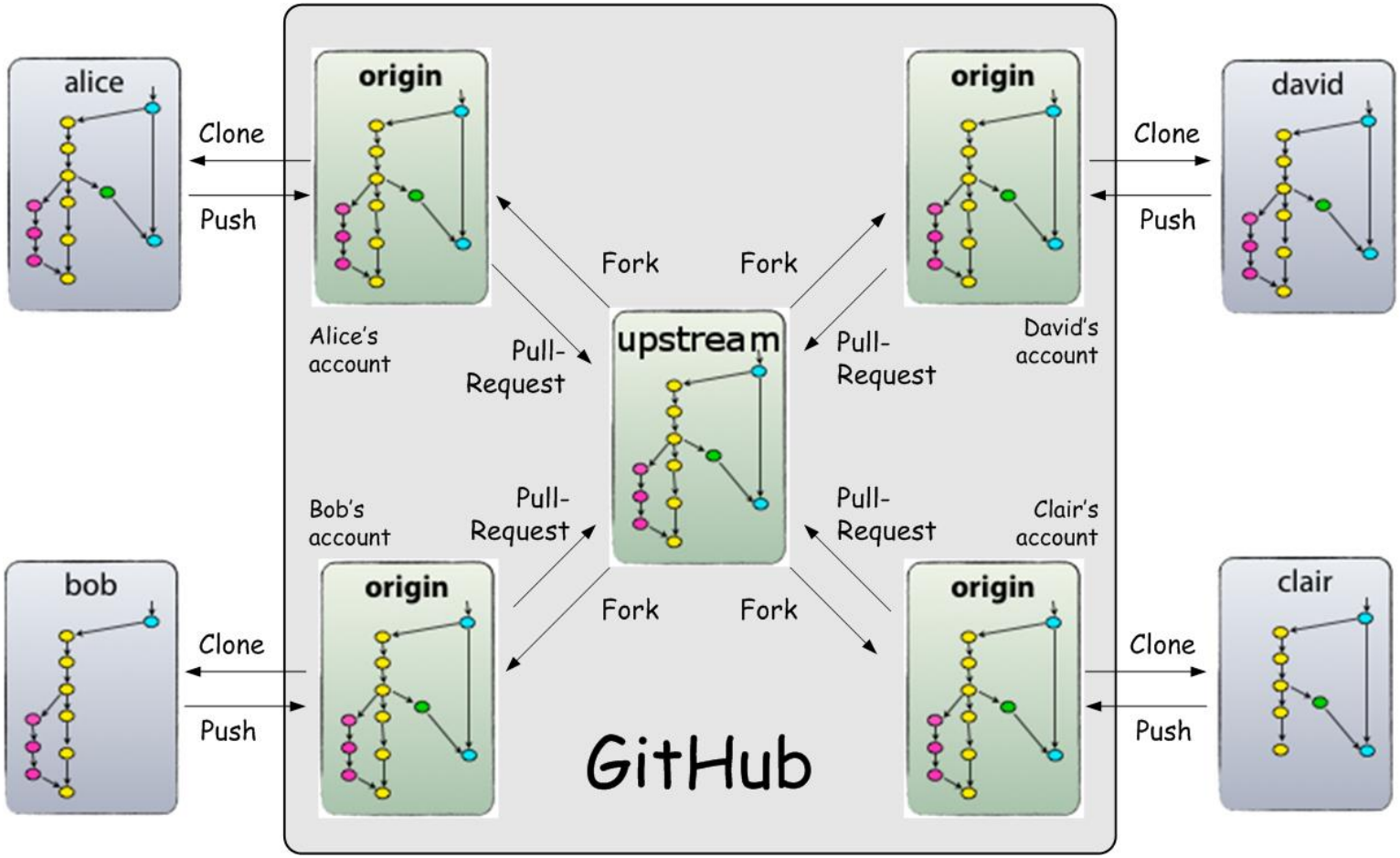
Extra: Visualisatie

- https://code.google.com/p/gource/

  Voorbeelden:
  https://www.youtube.com/watch?v=pOSqctHH9vY
  https://www.youtube.com/watch?v=a-gAoYapM8U

REFERENCES

Github: fork & pull-request



GIT COMMANDS

All commands

- Good cheatsheet: http://www.markus-gattol.name/misc/mm/si/content/git_workflow_and_cheat_sheet.png

Basic Commands - git

git – view all commands

$ git

Basic Commands - Init

Init - create an empty new repository

$ git init

Basic Commands - Status

Status - show differences between what has been committed and HEAD

$ git status

Basic Commands - Add

Add – add files to the stage

$ git add foo.info

Basic Commands - Commit

Commit – stores contents of the index in a commit along with a message

$ git commit –m “Added foo.info”

Basic Commands - Log

Log– view previous commits

$ git log

Basic Commands - Checkout

Checkout = checkout branches or previous commits

$ git checkout coolfeaturebranch

$ git checkout 1c899fed6ed

Remote - add

Add a repository that you track

$ git remote add origin git@github.com:brandonneil/test.git

Push

Update remote branch with changes from local branch

$ git push –u origin master

-u = add a tracking reference

Clone

Clone a repository into a new directory

$ git clone git@github.com:brandonneil/test.git

Pull

Fetch from and merge with another repository or local branch

$ git pull

EXERCISE

Practice

```
$ mkdir test
$ cd test
$ git init .
$ git status
```

1. Create a new directory
2. Move inside the new directory
3. Initialize the new directory as a git repository
4. Show the current status of the repository

Practice

$ notepad hello.txt

Type “Hello World” then save and exit

$ git status

git shows hello.txt as “untracked”

Untracked files are files which are in the current directory but *are not under version control!*

Practice

```
$ git add hello.txt
$ git commit -m “Add hello.txt”
$ git status
$ git log
```

1. Add hello.txt to version control
2. Commit changes in current repository (-m tells git to save a “commit message” with this commit)
3. Show status of repository
4. Show the commit log (a sort of history)

Practice

“Ammend” the last commit, telling git who you are:

$ git commit --ammend --author=”Tim dams<tim.dams@artesis.be>”

Make the settings permanent:

$ git config --global user.name “Tim Dams”
$ git config --global user.email tim.dams@artesis.be

Practice: braching & merging

```
$ git branch goodbye
$ git checkout goodbye
$ git status
```

1. Create a new branch called “goodbye”
2. Switch to the new branch
3. Print out the status of the repository on the current branch. Note the [goodbye] in the top left… we are on the “goodbye” branch!

Practice: braching & merging

```
$ gedit goodbye.txt
$ git add goodbye.txt
$ git commit -m “Add goodbye.txt”
$ git checkout master
$ git merge goodbye
```

1. Put text in goodbye.txt and then save and exit
2. Add goodbye.txt to version control
3. Commit changes to the current repository
4. Switch back to the master branch
5. “Merge” the changes from the “goodbye” branch into the current branch.