ACM AUTH
Student Chapter

GIT

Version Control System

Presented by Stefanos Laskaridis

SOME HISTORY

- Before Version Control Systems:
  - Email tarballs and use of **diff** and **patch** programs to find differences and apply them.
  - Really really impractical!

VERSION CONTROL

- A system that saves "snapshots" of a project's state that you can recall later.

- It's not restricted only to programmers.
  e.g. graphic design project, text, etc.

TIMELINE

tarballs of patches

CVCS (Centralized Version Control Systems)

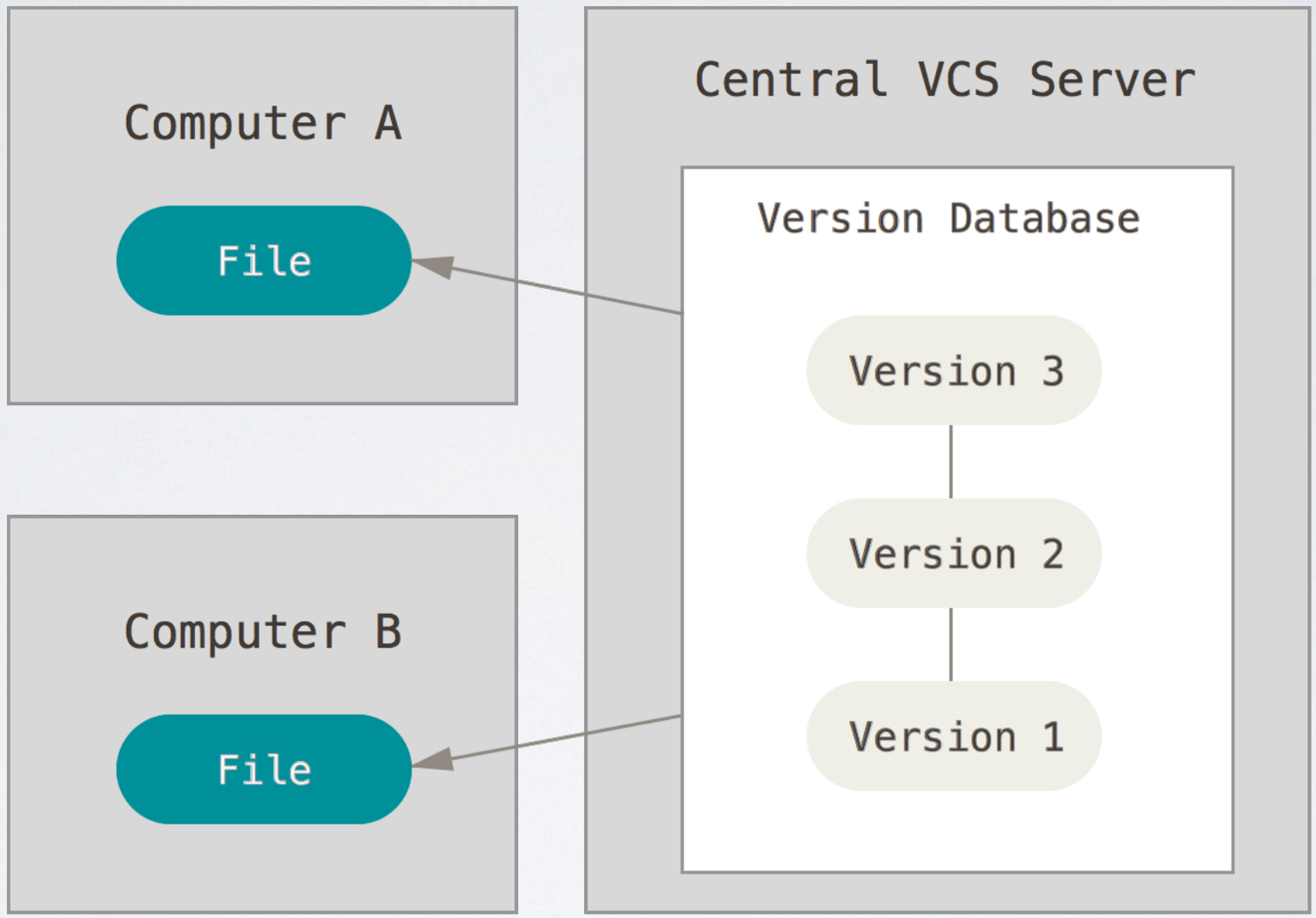
e.g. Subversion, Peforce, CVS

DVCS (Decentralized)

e.g. git, Mercurial, Bazaar

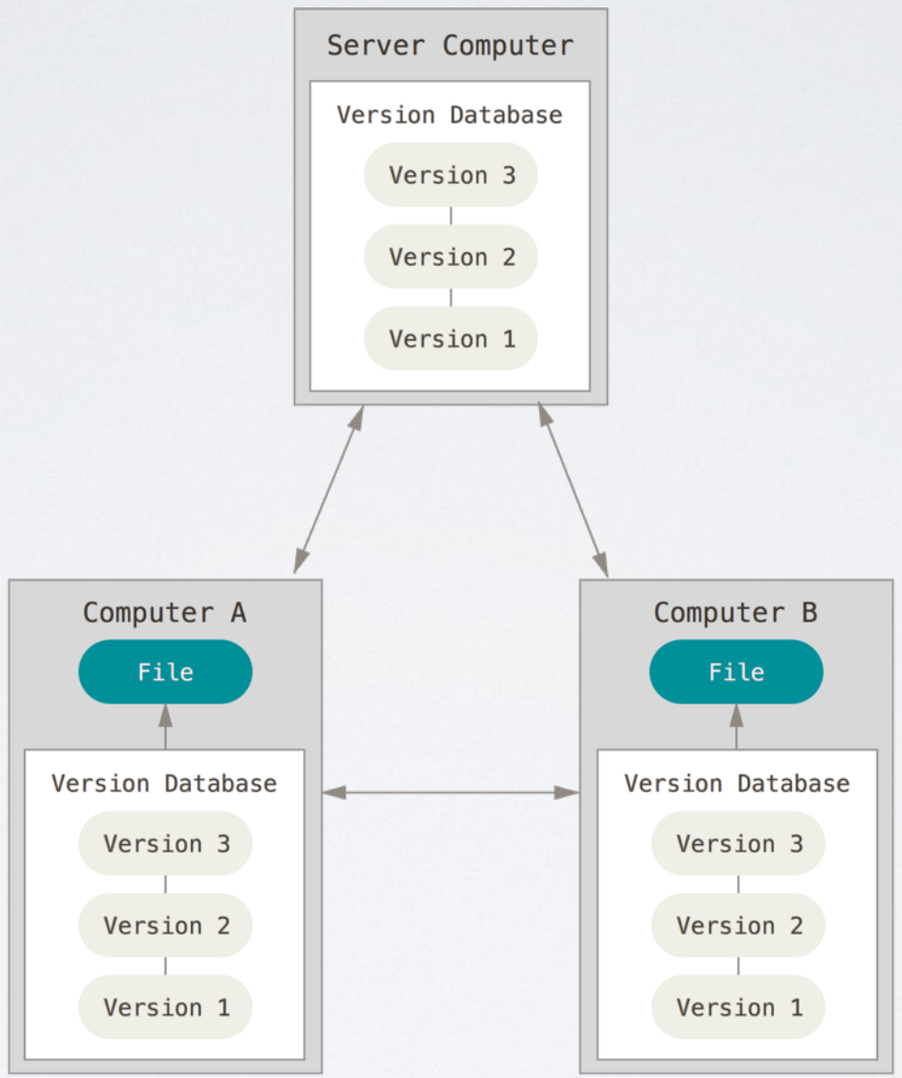
CVCS

Single Point of Failure



from the book "Pro git" by Scott Chacon and Ben Straub

DVCS



from the book "Pro git" by Scott Chacon and Ben Straub

SO WHAT IS GIT?

SOME FACTS

- Created by Linus Torvalds

- Written for the collaboration of the Linux Kernel development

- Originally written in C in a weekend
- Emphasises on speed

GIT PRINCIPLES

- Every computer has a full copy of the repository
- Git is designed for speed
- Git tracks snapshots of a miniature filesystem
- Git is designed for data integrity

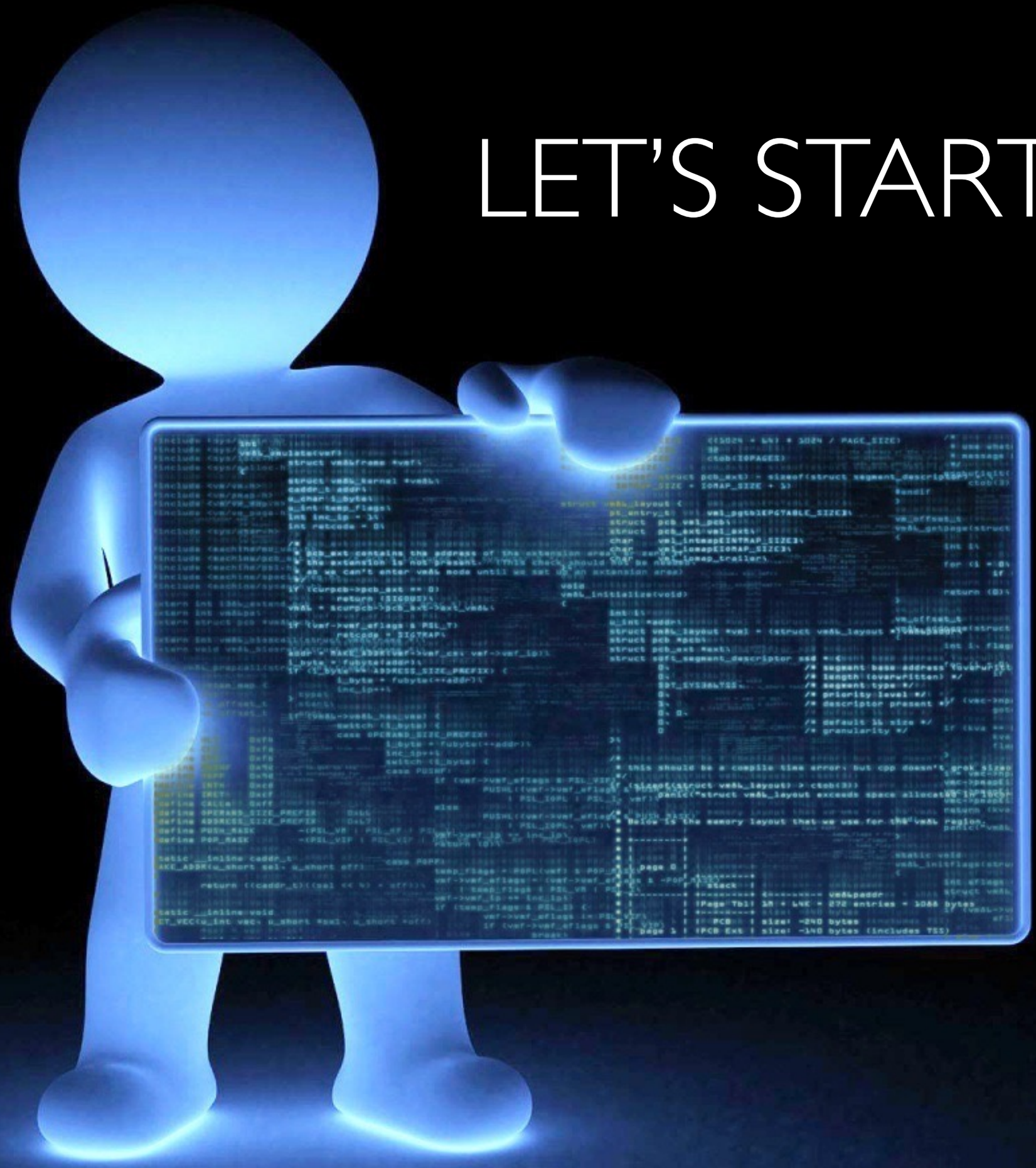
“THE STUPID CONTENT TRACKER”

TOOLS THAT WE ARE GOING TO NEED …

TOOLS THAT WE ARE GOING TO NEED

- A computer (dah!)
- Any major OS of your choice: Linux, Mac OS, Windows (preferably the first two)
- Installed git with Bash shell
- An account at a remote central repository (e.g. GitHub, BitBucket, etc.)
- Sourcetree or another git client (optional)

LET'S START



INSTALLATION

Linux	Windows	MacOS

`yum install git-core`
 	
`apt-get install git-core`
	http://code.google.com/p/msysgit/	http://code.google.com/p/git-osx-installer/

* There are GUIs as well, but we'll do it the "interesting" way

SOME JARGON

Term	Description
Repo (aka Repository)	Essentially a directory where there is code in it.
Commit	A commit, or "revision", is an individual change to a file (or set of files). It's like when you *save* a file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by who. Commits usually contain a commit message which is a brief description of what changes were made.
Remote	A version of a branch/directory/file that is hosted on another repository
Pull	Refers to when we are fetching changes to a repository and then merging them.
Push	Sending the committed changes in a repo to a remote repository.

FIRST SETUP

```
> git config --global user.name “Your name”

> git config --global user.email
“yourmail@domain.com”

> git config --global core.editor vim

# Check settings
> git config --list
```

Writes changes into ~/.gitconfig

GIT HELP

```
# 3 ways to get help

> git help <command>
# or
> git command --help
# or
> man git-<command>
```

FIRST STEPS

- Create a project in whatever language you want and put it in a directory of your choice

- Write some sample code

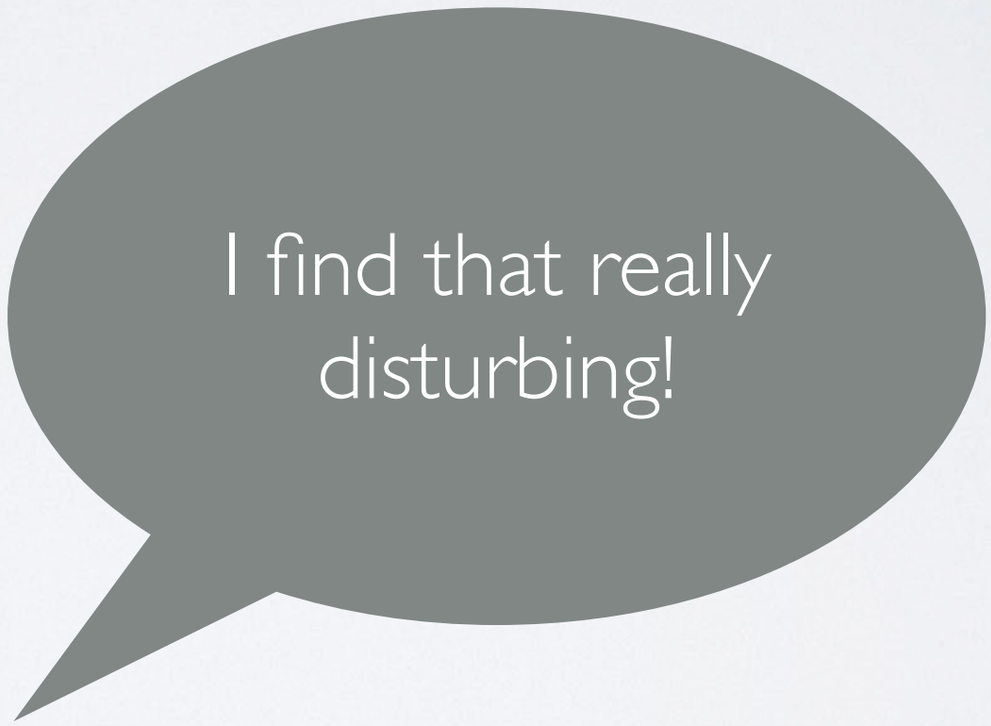
- Create a local repo

CREATE A LOCAL REPO

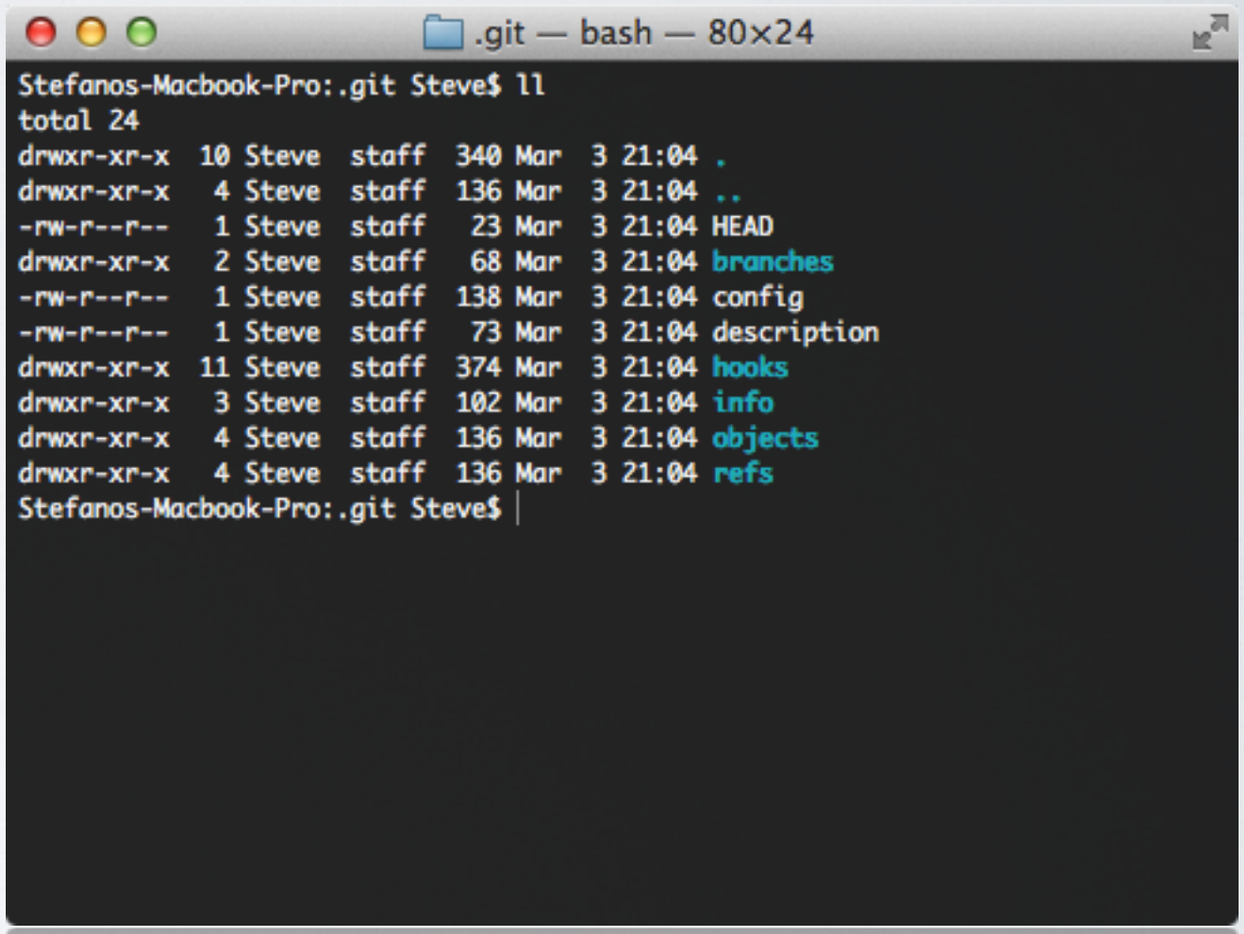
- Open a terminal window:
  - Navigate to the directory of your project

```
# create an empty repo
>git init

# add the current directory to the staging area
>git add .

# commit your staging area to the repo
>git commit -m “My first commit”
```

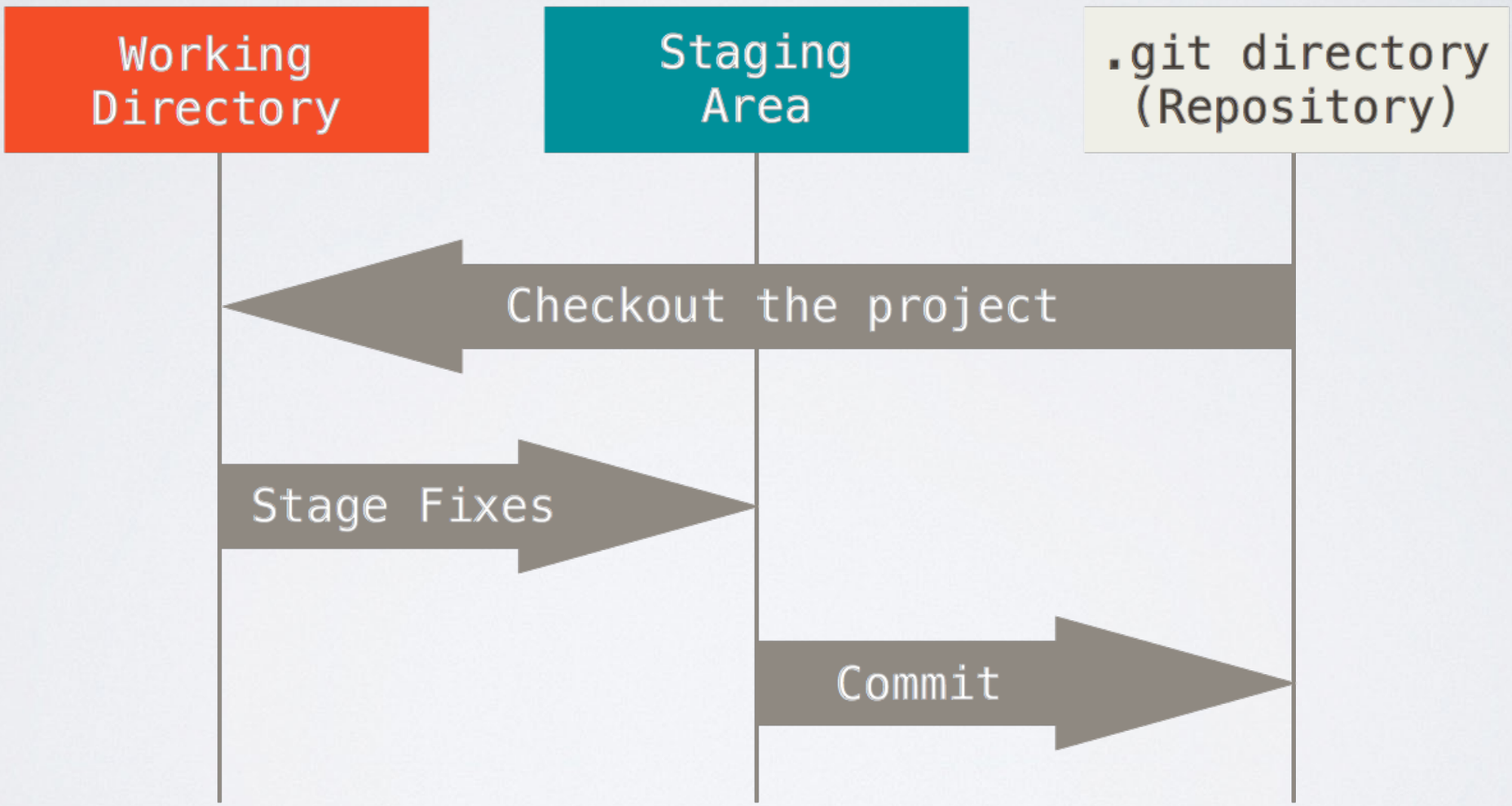
- If you miss the -m part, you will be presented with your core.editor and be asked to write a commit message.



- Now, you can see a folder named .git in your directory. (. stands for being “hidden”)



THE WORKFLOW



from the book "Pro git" by Scott Chacon and Ben Straub

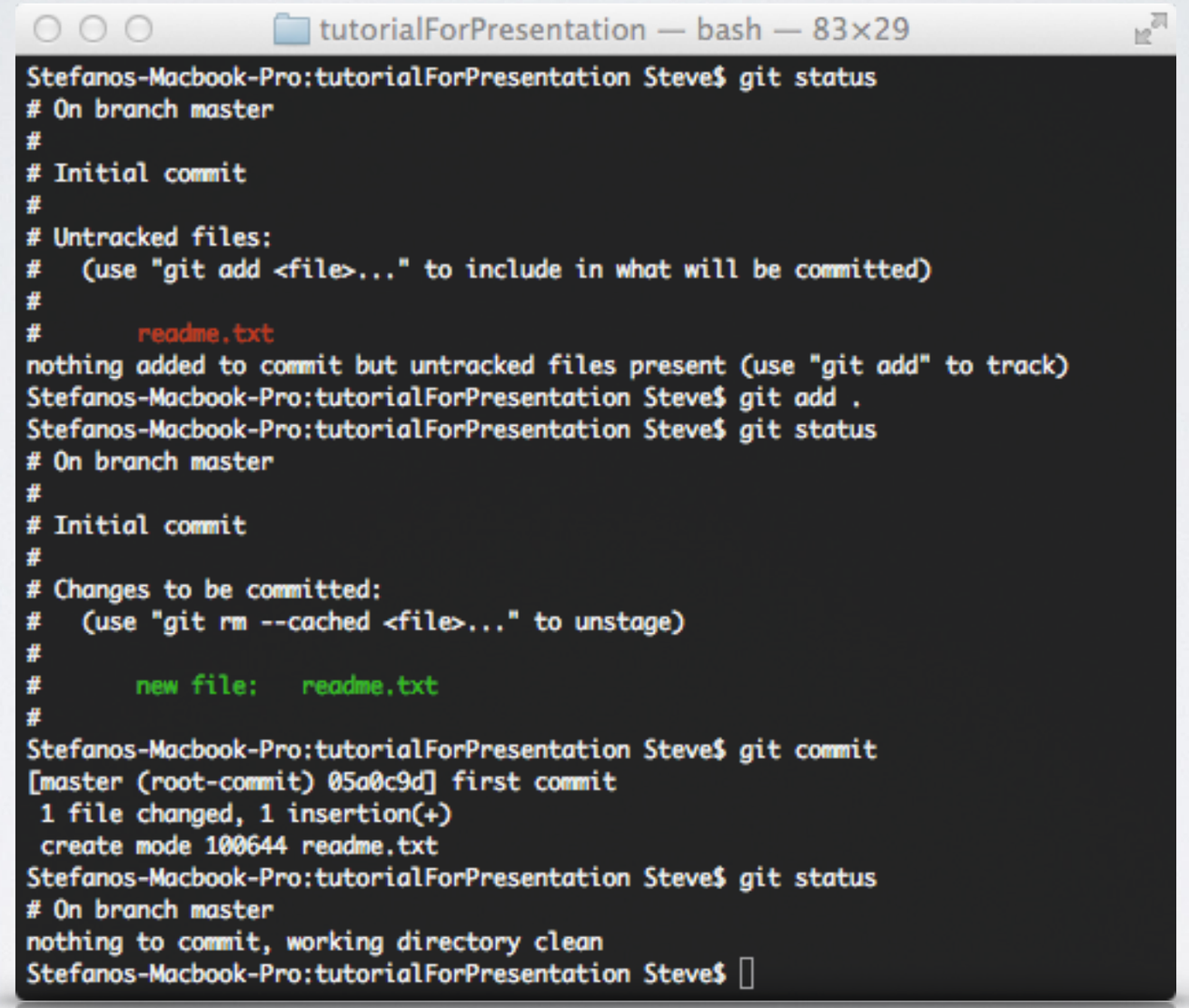
WHEN YOU MODIFY YOUR WORKING TREE …

- You add the selected files to the staging area

- You commit the changes

  - this creates a new persistent snapshot of the complete git repo

GIT STATUS


tutorialForPresentation — bash — 83×29
Stefanos-Macbook-Pro:tutorialForPresentation Steve$ git status
On branch master
#
Initial commit
#
Untracked files:
(use "git add <file>..." to include in what will be committed)
#
readme.txt
nothing added to commit but untracked files present (use "git add" to track)
Stefanos-Macbook-Pro:tutorialForPresentation Steve$ git add .
Stefanos-Macbook-Pro:tutorialForPresentation Steve$ git status
On branch master
#
Initial commit
#
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
#
new file: readme.txt
#
Stefanos-Macbook-Pro:tutorialForPresentation Steve$ git commit
[master (root-commit) 05a0c9d] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
Stefanos-Macbook-Pro:tutorialForPresentation Steve$ git status
On branch master
nothing to commit, working directory clean
Stefanos-Macbook-Pro:tutorialForPresentation Steve$

SEE CHANGES

- Use git diff to see the changes made to your tracked files

- You can specify a filename to see specific changes

IGNORING FILES

- Create a text fi
- In this file, ; es you want to be untracke

If you want added files to be removed after you add your .gitignore, type:
git rm -r --cached

- **Example:** Ignore all html files

```
> touch .gitignore
> echo *.html >> .gitignore
> git add . ;git commit -m “your message”
```

IGNORING DIRECTORIES

- By default, git does not track empty directories
- If you want so, just put an empty **.gitkeep** text file in the folder

GIT CLONE

```
# clone the entire repository to a local directory
> git clone <url>

# clone the central repo
> git clone https://github.com/GNULinuxACMTeam/
gitworkshop.git
```

GIT LOG

- A powerful tool in the hands of the programmer
- Helps see the changes made at any stage of the repo.
- There is also a variation: git shortlog

GIT LOG

```
> git log
> git log --oneline
> git log --pretty
> git log --pretty=oneline --max-count=2
> git log --pretty=oneline --since='5 minutes ago'
> git log --pretty=oneline --until='5 minutes ago'
> git log --pretty=oneline --author=<your name>
> git log --pretty=oneline --all
> git log --all --pretty=format:'%h %cd %s (%an)' --
since="7 days ago" - formatting the output
> git log --pretty=format:'%h %ad | %s%d [%an]' --
graph --date=short - ultimate log format
```

You can also set hist = git log ... as an alias in ~/.gitconfig

GIT LOG

Option	Description
-(n)	Show only the last n commits
--since, --after	Limit the commits to those made after the specified date.
--until, --before	Limit the commits to those made before the specified date.
--author	Only show commits in which the author entry matches the specified string.
--committer	Only show commits in which the committer entry matches the specified string.
--grep	Only show commits with a commit message containing the string
-S	Only show commits adding or removing code matching the string

GIT LOG PRETTY FORMAT SPECIFIERS

Option	Description
%H %h	Full hash, abbreviated commit hash
%an	Author Name
%ae	Author email
%ad	Author date
%s	Subject
%Cred, %Cgreen, %Cblue, %Creset	Coloring the output
%n	Newline

And many more ...

REVERT TO A PREVIOUS VERSION

```
> git checkout <hash>
> git checkout <tag>
```

WHAT IS A TAG?

GIT TAGS

- Tags are named commits

- Instead of referring to them with their hash, you put a name on them

- It's usually used for versioning (annotated tags)

GIT TAGS

```
# Show all tags
> git tag

# Tag the current version
> git tag <tag_name>
> git tag <tag_name> -m <message> # annotated tag

# Tag a specific commit
> git tag <tag_name> <hash>

# Delete tag
> git tag -d <tag_name>
```

TRY IT YOURSELF

TO DO

- Make some changes to the repo (add a new file or edit existing ones)
- Commit your changes
- Tag your last commit with name 0.0.2
- Checkout a past commit (find the hash from the history)
- Check the status of the checked out repostitory
- Return to master

REVERT CHANGES TO A FILE

You can see the second command in git status when having uncommitted changes in the staging area.

```
# Reverting changes to a file
> git checkout <filename> # resets an uncommitted file
> git reset HEAD <filename> # resets the file in the staging area
```

UNDOING COMMITS

UNDOING COMMITS

Sometimes you realize that something that you have committed is wrong and wish to undo that commit. Not intended for bug fixes.

```
# Return to the state before your last commit, puts it
into history
> git revert HEAD
> git revert --no-edit

# Remove commits from a branch
> git reset <hash>

# Amending last commit
git commit --amend -m "commit message"
git commit --amend --author "name
<email@address.com>"
```

GIT RESET

- git reset essentially changes the HEAD pointer to a previous commit

- git reset <hash>

  --soft: Change the HEAD, leave the index and files unchanged

  --mixed: Change the HEAD and the index, leave the files unchanged

  --hard: Change the HEAD, the index and the files

(RE)MOVING FILES

```
> git mv <path_to_file_1> <path_to_file_2>
# or
> mv <path_to_file_1> <path_to_file_2>;git rm
<path_to_file_1>;git add <path_to_file_2>
```

BARE REPOSITORIES

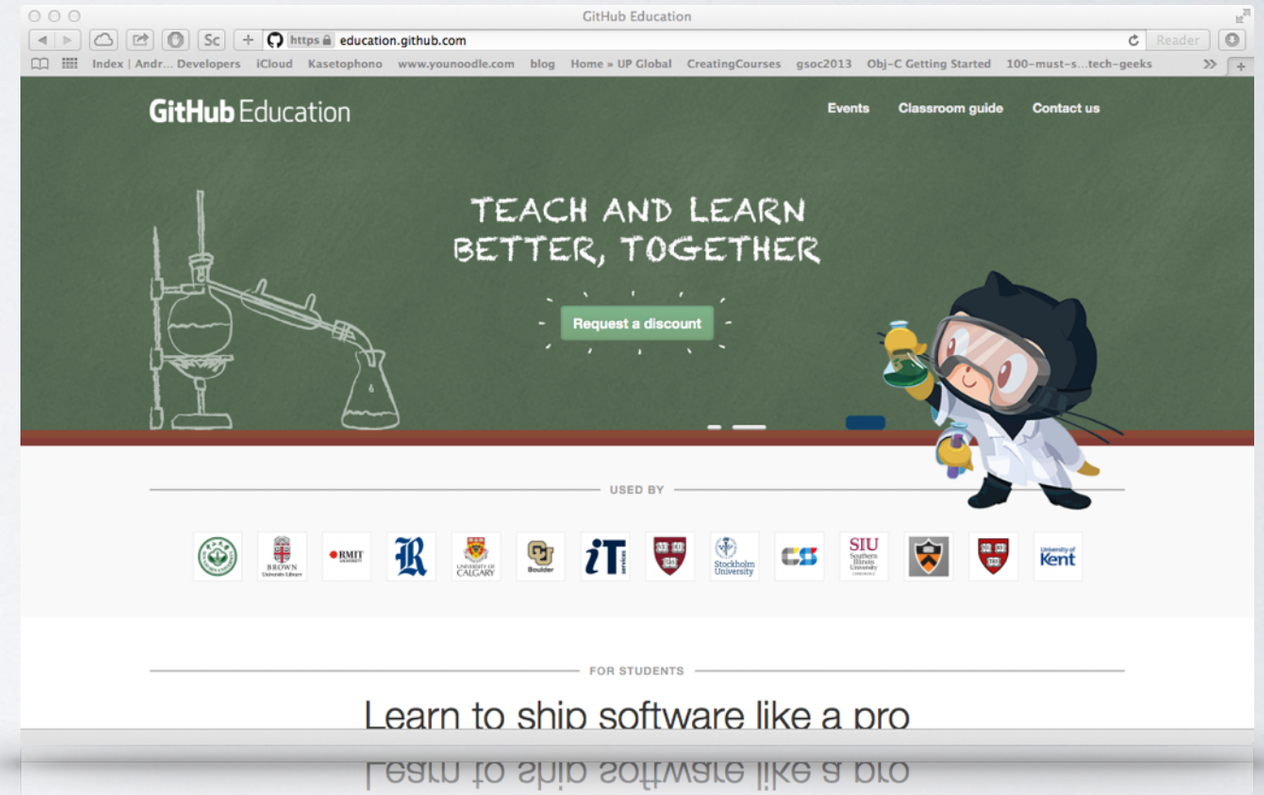
- Distinction between bare and working repositories:
  - **Bare repositories** only have the *.git folder
  - **Working repositories** have the working tree and the *.git folder

BARE REPOSITORIES

- Bare repositories are usually used for sharing
- The convention is that repositories ending in ".git" are bare repos

GITHUB FOR EDUCATION

- Github offers premium account to university students.
- Add you university mail at your github account.
- Go to education.github.com
- Apply for the premium account using your university mail
- There will be a reply accepting your request.



GITHUB OFFERS TO STUDENTS

- Atom (already open-source)
- Bitnami Business 3 plan (normally 49$/month)
- Crownflower (normally 2500$/month)
- DigitalOcean (100$ credit)
- DNSimple Bronze plan (normally 3$/month)
- Github Micro Account (normally 7$/month)
  5 private repos
- Hackhands (25$ credit)

GITHUB OFFERS TO STUDENTS

- Namecheap (.me domain and 1yr SSL certificate, normally 8.99$/yr)
- Orchestrate (developer account, normally 45$/month)
- Screenhero (normally 9.99$)
- SendGrid
- Stripe (waived first 1000$ fees)
- Travis CI (private builds, normally 69$/month)
- Unreal Engine development tools (normally 19$/month)

GOING REMOTE

CLONE A REPO

When cloning a repo, the origin remote link is created by default.

- Let's clone our repo in another directory:

```
# We are in the directory of our repo
> cd ..
> mkdir “cloned repo”
> cd “cloned repo”
> git init
> git clone ../repoDirectory
# a directory called “repo directory” is now created in the working directory
```

WORKING WITH A REMOTE REPOSITORY

- After you have created a centralised repo in a remote server (e.g. github), you have to take the repo locally.

- How?

CLONING A REMOTE REPO

- To get a remotely created repo, you can use the protocols used by the service to connect to the repo. Usually, these include:
  - http(s) (hopefully)
  - ssh
  - git

REMOTE REPOS

- Push changes to another repo
  - git push <remote url>
- Fetch from and integrate with another repository or a local branch.
  - git pull
  - git fetch (brings the changes to the remote repo, but does no integration)
  - git merge origin/master (merge these changes)

BRANCHES

- Named pointer to commit
- You can work on different branches independently from each other
- By default, you are committing to the master branch of the project
- Each local clone of a remote repo is a different branch
- You can create, delete, or merge different branches in one repository

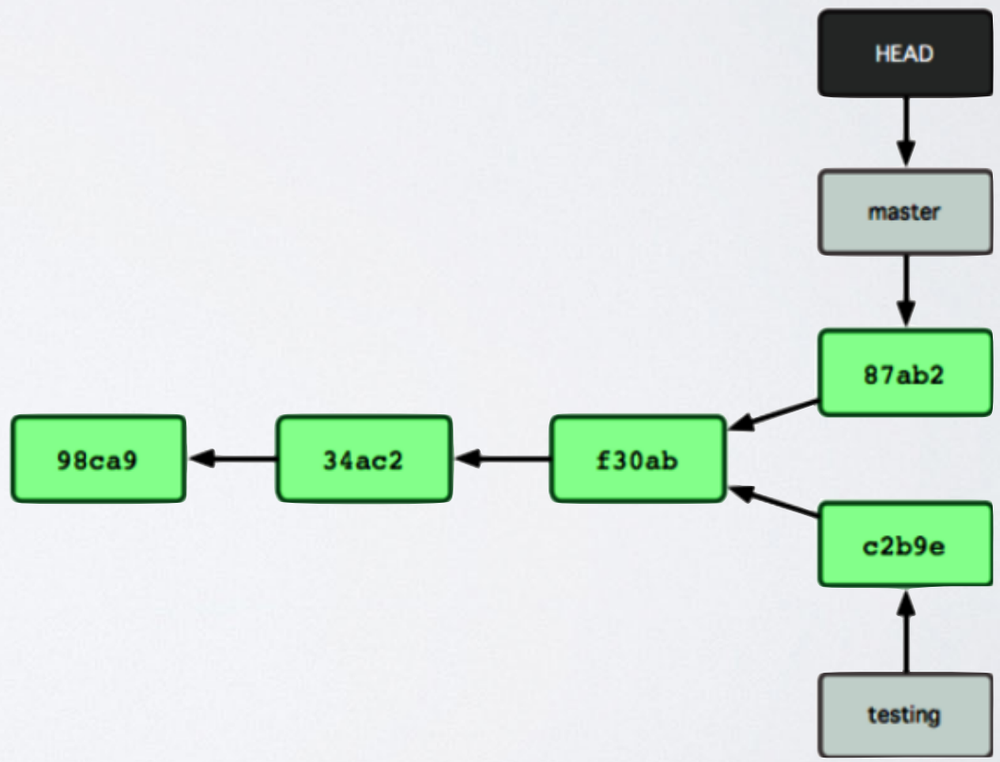
BRANCHES

```
# see all the available branches
> git branch -av

# list branches and tags in a remote
pointer called origin
> git ls-remote origin

# create new branch
> git branch <branch_name>
> git checkout -b <branch_name>

# rename a branch
> git branch -m <old_name>
<new_name>
```

```
#delete a branch
> git branch -D <branch_name>

# switch to a branch
> git checkout [branchname]

# see differences between branches
> git diff [branch1] [branch2]
```

DIVERGED BRANCHES

- This happens when a branch has different commits that another. Usually, the working directory has changed too.



Graph taken from git-scm.com

- Imagine that you and a fellow programmer are contributing to the same repository, asynchronously.

- Imagine also that you both change the same part of code.

- What's going to happen?

A CONFLICT :P

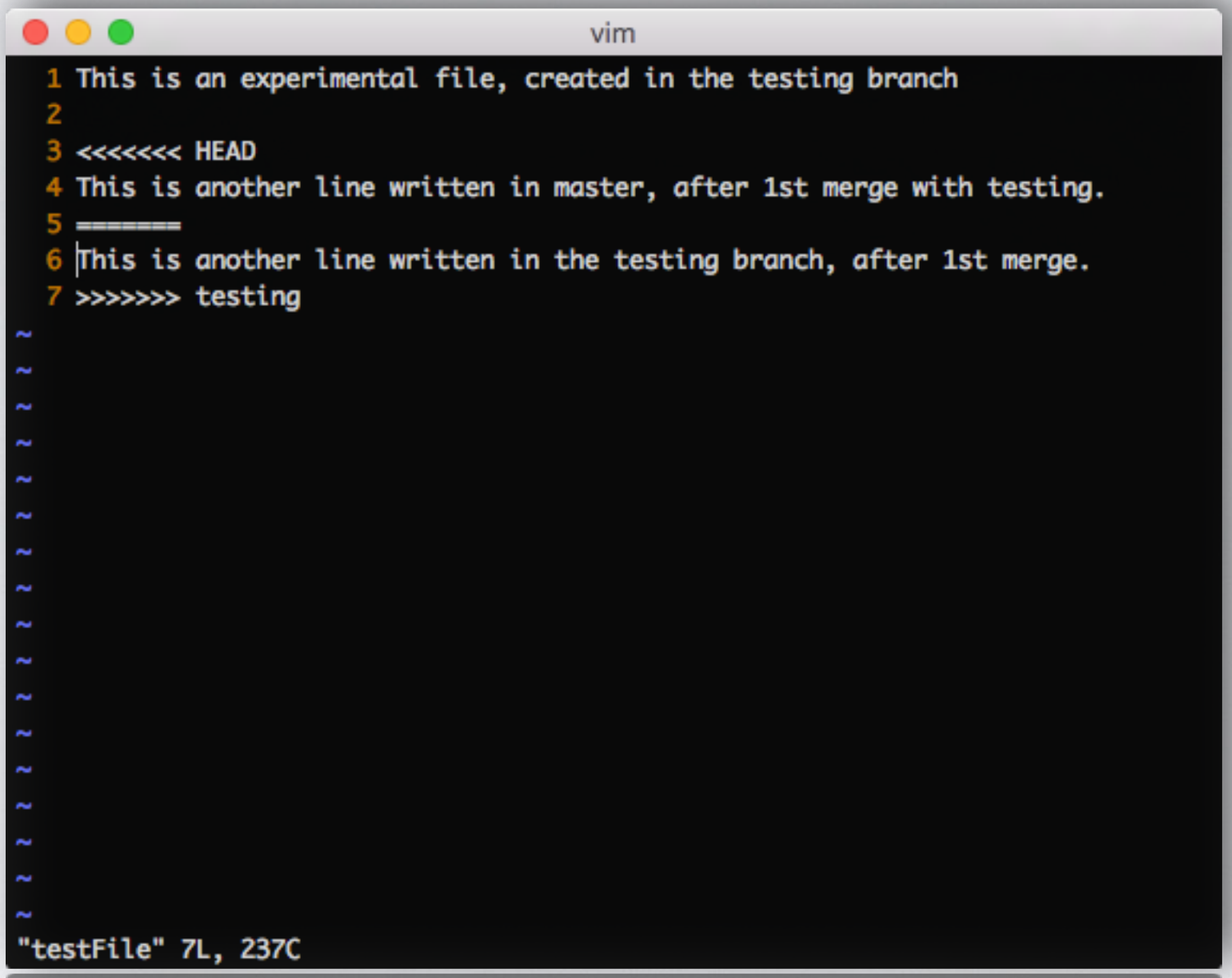
CONFLICT RESOLUTION

- When a conflict arises, git puts some lines inside the file having conflicts in order to show you where the problem exists.

- You have to solve the problem manually, or semi-automatically by choosing which part of code you want.

- Then you commit your changes and the two branches (yours and the remote) will merge.

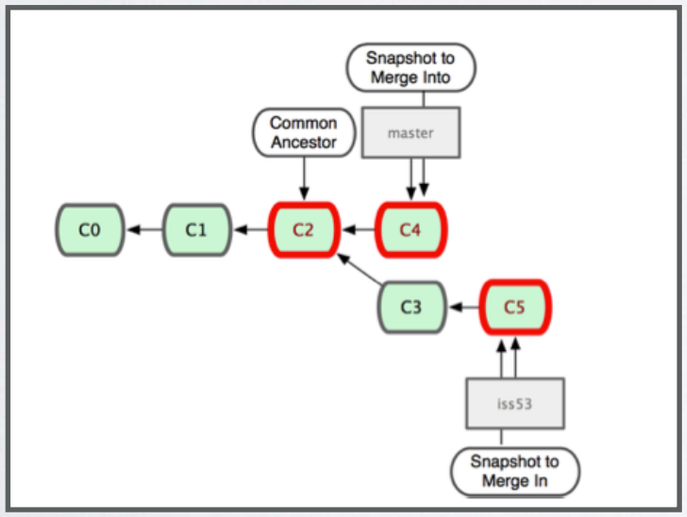
HOW GIT SHOWS YOU CONFLICTS



OPERATIONS ON BRANCHES

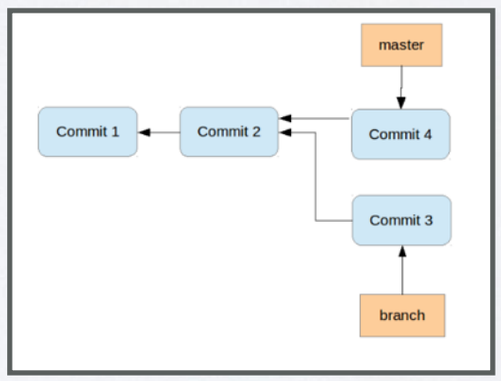
Merge Branches

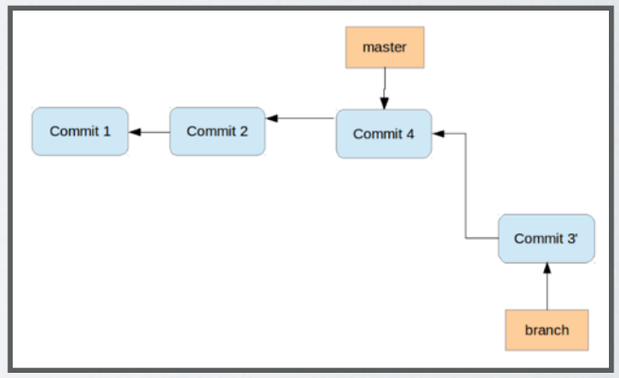
```
> git merge <hash>
```



Rebase Branches

```
> git rebase
```





MERGING

```
# Add a default merge tool
echo "git config --global merge.tool vimdiff" >> ~/.gitconfig
```

REBASING

- With the rebase command, you can take all the changes that were committed on one branch and replay them on another one.

GIT STASH

- Useful when you want to move aside the current things open in your working directory and work on something else.

- It's like putting them under the carpet ...

GIT STASH

```
# push to stash
> git stash

# pop from stash
> git stash pop

# show stashes
> git stash list

# create new branch from stash
> git stash branch <branchname> [<stash>]
```

USEFUL INFORMATION

INTERESTING TOPICS FROM HERE …

- Solving merging collisions & merging strategies
- Hosting your Git repositories
- Reverting Committed Changes
- Finding Buggy commits (git bisect)
- git internals
- Continuous Integration

GIT ALTERNATIVES

Mercurial	Bitbucket, Go Programming Language, Mozilla, Octave, OpenOffice
SVN (Subversion)	Tends to get deprecated
CVS (Concurrent Versions System)	The first SCM supported by Sourceforge
Bazaar	Distributed revision control sponsored by Canonical
Perforce	Popular among game developers. Free for up to 20 users.

LINKS

Git reference

Chacon, Straub - Pro Git

Git Immersion

Git Tutorial by vogella.com

Linus Torvalds on Git

github.com

education.github.com

bitbucket.org

SourceTree

GUI Clients

ACM AUTH
Student Chapter

THANK YOU