

## Agenda

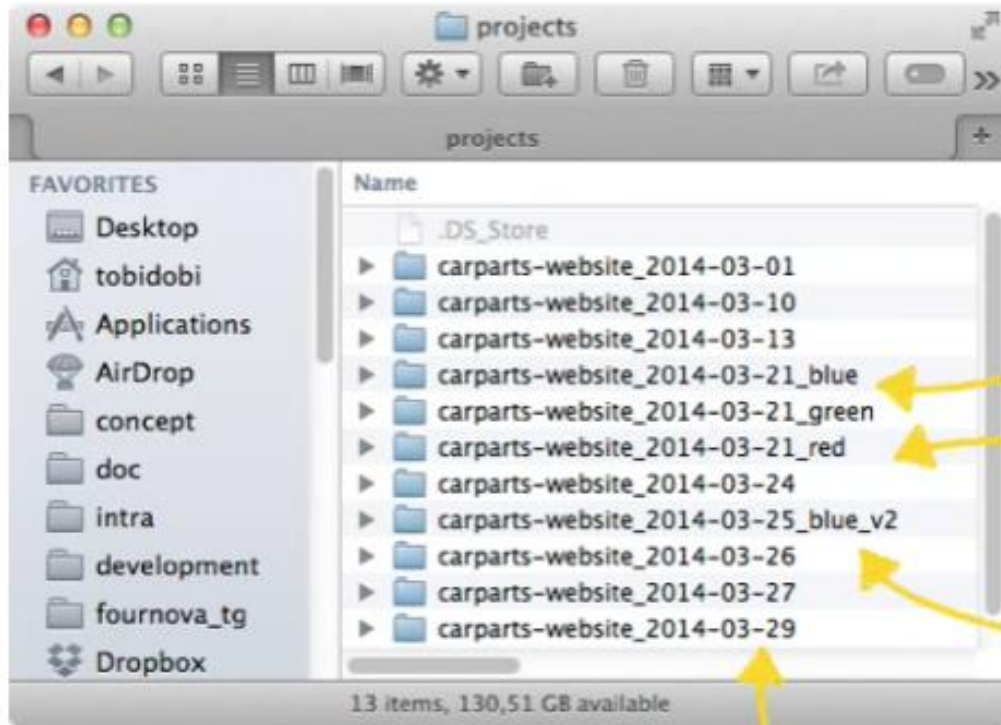
1. The need for Version Control
2. Using git for version control

## **Scenario:**

- You are a developer writing and testing programs
- Your programs are part of a larger application development project
- Other developers are working on this application too

Where do I store my code?

# Version Control



What **exactly** was changed in each version?

...and what do these changes **mean**?  
Did anyone document / comment them?

Why do we store the **whole project**, instead of just the modifications?!

How do you keep variants of the project in **sync** while it moves on?

Using a simple file system to track our code




## **Problems:**

- Losing History
- Someone else can overlay my code
- What if I get YOUR bugs in my code
- My code was working. Now it's broken. What happened?
- I fixed that last week. What happened to my change?
- What if two of us change the same file at the same time?

BUT – I need a private workspace to be productive, and I'd like a private copy of the entire repo

# Version Control

Table 5.1. Lines of Communication

People	1	4	10
			
Lines	None	6	45

The most productive developer is alone. A solitary developer never has to worry about coordinating with anyone else. But as soon as the project goes plural, there is overhead.<sup>1</sup> And for every developer added to the team, the overhead gets worse.<sup>2</sup>

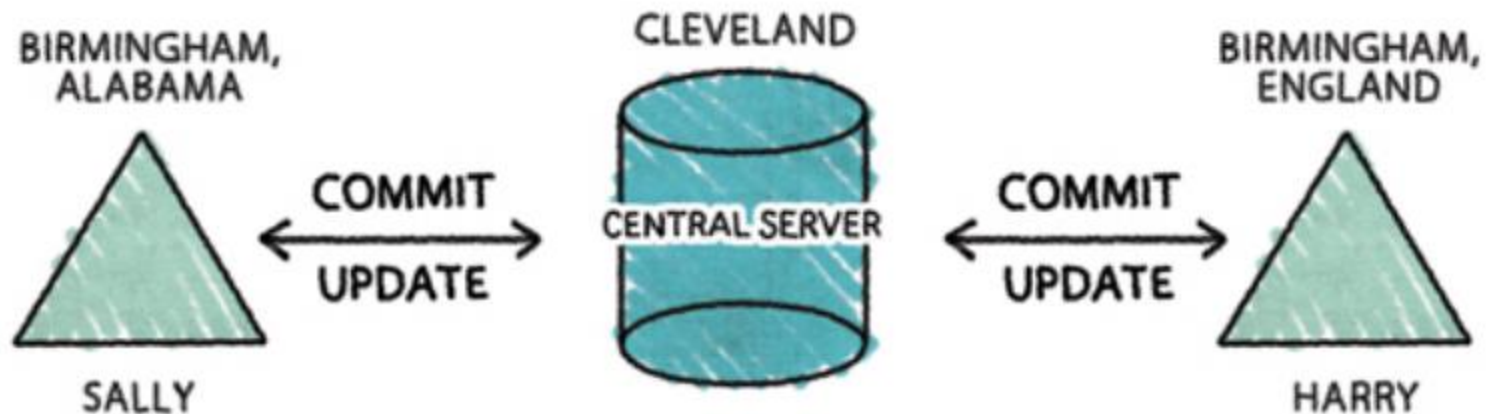
<sup>2</sup> The function is  $n*(n-1)/2$ .

## **I want a REPOSITORY:**

- Coordinates across multiple developers on a team
- Keeps History
- Allows me to revert back to a prior version of my code.
- Allows me and others to work on the same module at the same time.
- Tracks WHO is working on WHAT
- Prevents us from clobbering each others' updates.
- Describes each version of each module.
- Allows “check out”, “check in”
- Allows merge management
- Can track changes and compare one version to another
- Provides a log of all changes
- Allows me to add, delete, change, rename code

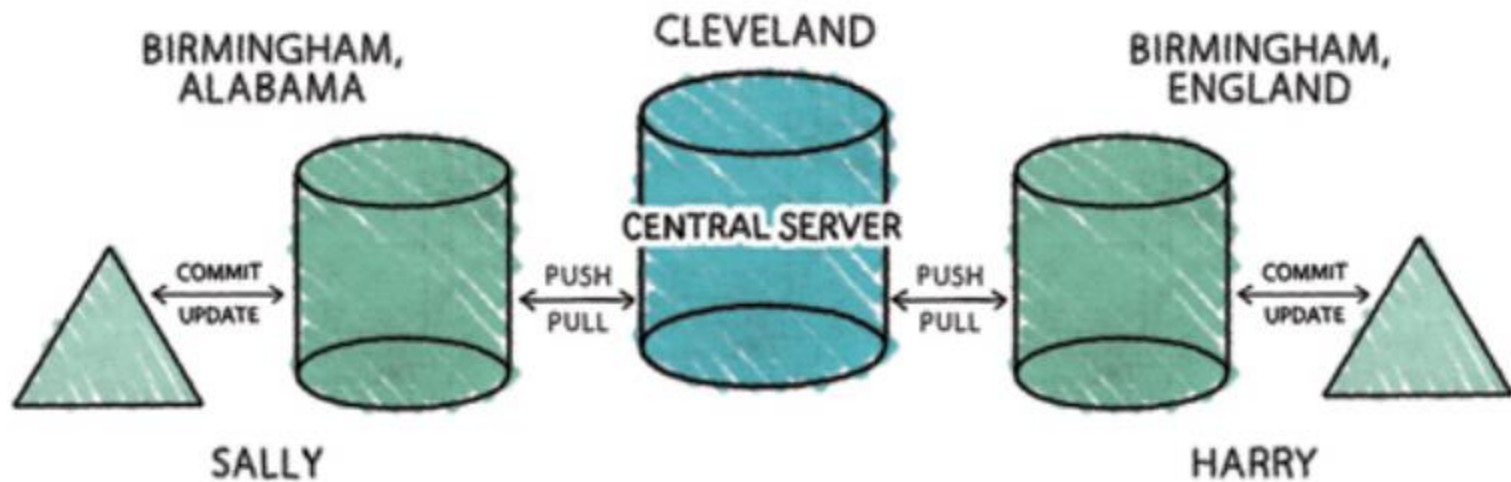
- **A “Version Control System” provides these features**
- **Two designs:**
  - Central Repository “VCS”
  - Distributed Repository “DVCS”

## Central Repository “VCS”





## Distributed Repository “DVCS”



## **Why DVCS?**

- **Cheap, fast local branches (offline)**
- **But, Full local branches, with history**
- **History of changes, pulls, pushes, commits, merges**
- **Offline commits**
- **Each working copy is a full backup of the repo**

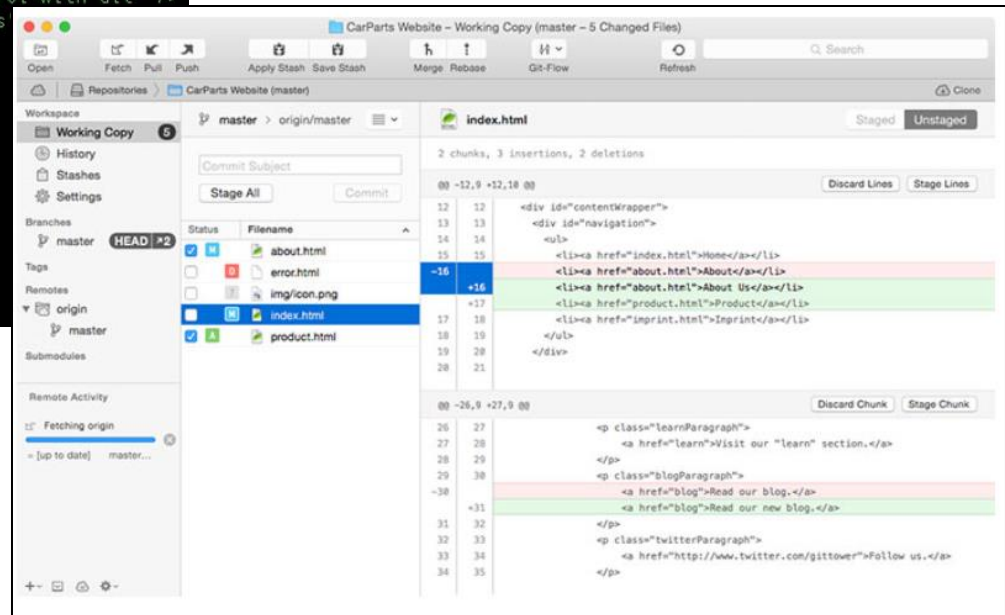
# Version Control

## Options

- Command line versus GUI tool

```
diff --git a/index.html b/index.html
index 56067f6..d15f148 100644
--- a/index.html
+++ b/index.html
@@ -4,6 +4,8 @@
<title>Tower :: Home</title>
<link rel="shortcut icon" href="img/favicon.ico" type="image/x-icon" />
<link type="text/css" rel="stylesheet" href="css/general.css"/>
+ <meta name="description" content="Learning version control with Git" />
+ <meta name="keywords" content="git, version control, vcs" />
</head>

@@ -12,8 +14,6 @@
<div id="contentWrapper">
  <div id="navigation">
    <ul>
-      <li><a href="index.html">Home</a></li>
-      <li><a href="about.html">About</a></li>
      <li><a href="imprint.html">Imprint</a></li>
    </ul>
  </div>
```



## **Popular Open Source VCS tools: (Central Repo)**

- **Concurrent Versions System (cvs)**
- **CVSNT**
- **OpenCVS**
- **Subversion**
- **Vesta**

## Popular Open Source DVCS tools: (distributed repo)

- **ArX**
- **Bazaar**
- **BitKeeper** – was used in Linux kernel development (2002 – April 2005) until it was abandoned due to being proprietary. It was open-sourced in 2016 in an attempt to broaden its appeal again.
- **Codeville**
- **Darcs**
- **DCVS** – decentralized and CVS-based
- **Fossil**
- **Git** – written in a collection of Perl, C, and various shell scripts, designed by Linus Torvalds based on the needs of the Linux kernel project; decentralized, and aims to be fast, flexible, and robust
- **GNU arch**
- **Mercurial** an Open Source replacement to Bitkeeper
- **Monotone**
- **SVK**
- **Veracity**

## **Popular free GUI tools (for git)**

<https://git-scm.com/download/gui/linux>

# History



1972: Source Code Control System ([SCCS](#))



1982: Revision Control System ([RCS](#))



1990: Concurrent Versions System ([CVS](#))



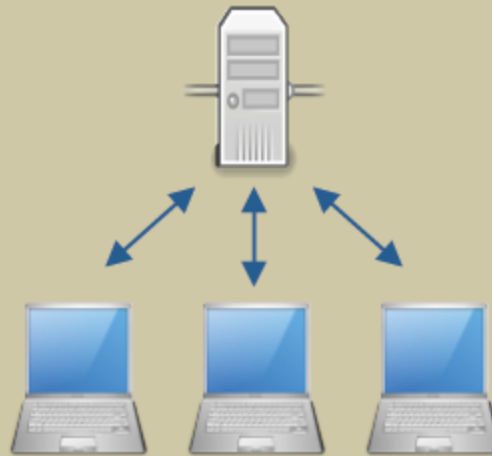
2000: Subversion ([SVN](#)), [BitKeeper](#)



2005: [git](#), Mercurial ([hg](#)), Bazaar ([bzzr](#))



# Local vs Client/Server vs Distributed



Local Operation

Individual Developer

No Locking

Connected Operation

Limited Team Size

Lock-Based



Disconnected Operation

Scalable Team Size

Branch-Based

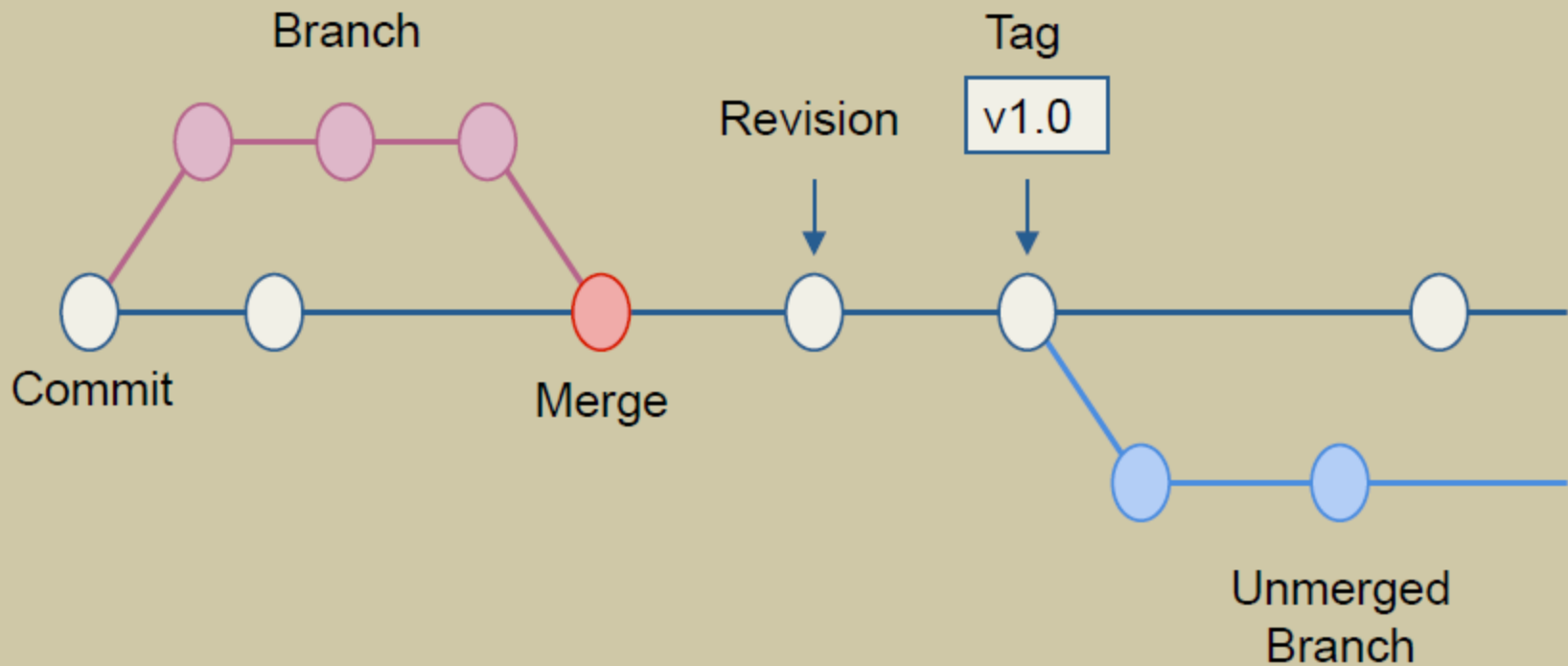




# More Terminology

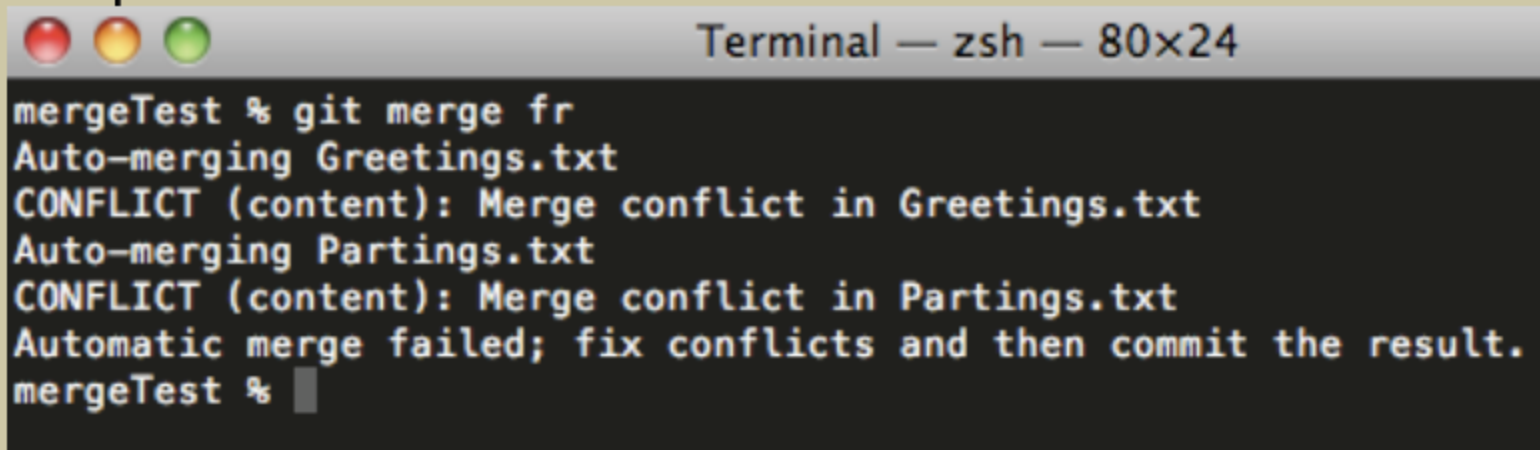
- Repository** : Version controlled collection of files/code
- Commit** : Write/Save changes to repository
- Revision** : A specific version of a repo/file
- Tag** : A logically named/labeled revision
- Branch** : A linear subset of changes within a repo
- Merge** : To combine discrete branches
- Diff** : The set of changes between two revisions

# Version Control – History tree



# Update Conflicts

- Conflicts happen when developers edit the same code and it cannot be automatically merged.
- In practice, can be reduced by adequate communication and coding strategy.
- Requires manual intervention.

A terminal window titled "Terminal — zsh — 80x24" with three colored window control buttons (red, yellow, green) in the top left. The terminal text shows a user at the "mergeTest" prompt running "git merge fr". The output indicates an automatic merge of "Greetings.txt" and "Partings.txt", but both result in "CONFLICT (content): Merge conflict". The terminal concludes with the message "Automatic merge failed; fix conflicts and then commit the result." and returns to the "mergeTest %" prompt with a cursor.

```
mergeTest % git merge fr
Auto-merging Greetings.txt
CONFLICT (content): Merge conflict in Greetings.txt
Auto-merging Partings.txt
CONFLICT (content): Merge conflict in Partings.txt
Automatic merge failed; fix conflicts and then commit the result.
mergeTest %
```

# Conflict Resolution

- Merge tools (text-based or GUI based)
  - Often the same tool as Diff
- Line-by-line comparison;  
allows user to choose which line(s) to keep.
- Once complete, “Mark as Resolved” and commit.

# Merge Example

The screenshot displays the TortoiseMerge application window titled "HelloClass.cs.mine - TortoiseMerge". The interface shows three panels: "Theirs - HelloClass.cs.r16", "Mine - HelloClass.cs.mine", and "Merged - HelloClass.cs".

**Theirs - HelloClass.cs.r16:**

```
9 {  
+10 ----public String SayHiTo {get; set;}  
+11 ----  
12 ----public String SayHello() {  
13 ----{  
14 ----String theGreeting;  
+15 ----theGreeting = "Hello SVN from Windows Forms!";  
+16 ----theGreeting = "Hello SVN from Windows Forms.";  
+17 ----theGreeting += " And Hello Ron.";  
17 ----return theGreeting;  
18 ----}  
19 }  
20 }
```

**Mine - HelloClass.cs.mine:**

```
9 {  
10 ----public String SayHello() {  
11 ----{  
12 ----String theGreeting;  
+13 ----theGreeting = "Hello SVN from Windows Forms!";  
+14 ----theGreeting = "Hello SVN from Windows Forms. Hello";  
14 ----return theGreeting;  
15 ----}  
16 }  
17 }
```

**Merged - HelloClass.cs:**

```
9 {  
+10 ----public String SayHiTo {get; set;}  
+11 ----  
12 ----public String SayHello() {  
13 ----{  
14 ----String theGreeting;  
-15 ----theGreeting = "Hello SVN from Windows Forms!";  
+16 ----theGreeting = "Hello SVN from Windows Forms. Hello";  
17 ----return theGreeting;  
18 ----}  
19 }  
20 }
```

The status bar at the bottom indicates: "For Help, press F1. Scroll horizontally with Ctrl+Scrollwheel. Left View: UTF8 BOM CRLF / - 1 / + 4. Right View: UTF8 BOM CRLF / - 1 / + 1. Conflicts: 2. CAP NUM SCRL".

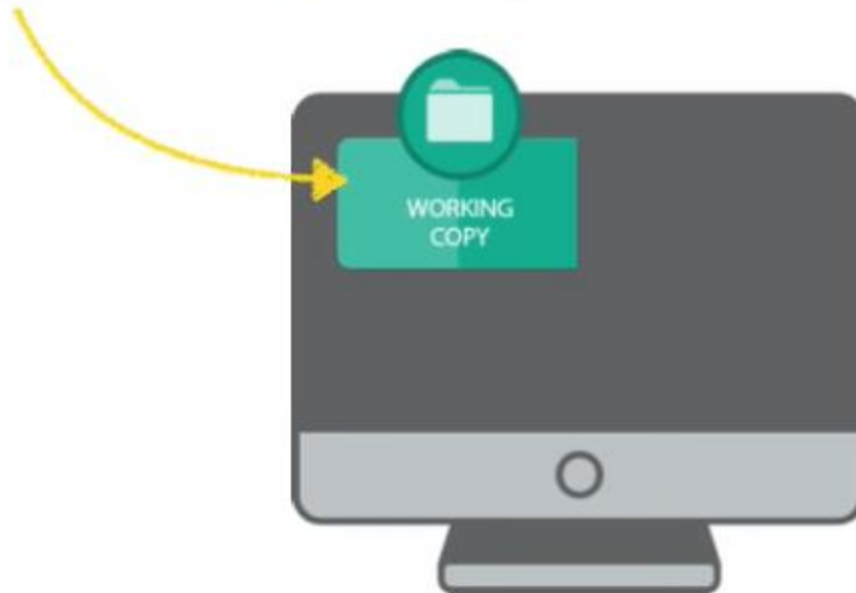
## **Best Practices**

- Do**
  - Commit Early, Commit Often
  - Use Clear Commit Messages
  - Communicate !! (resolving merge conflicts)
  - Only commit related changes in one “commit”
  - One topic per branch
  - Only commit source code
- Don't**
  - Commit Compiled Files
  - Commit Secure Info (e.g. credentials, passwords)
  - Merge Broken Code (i.e. don't break the master branch)
  - Combine multiple topics per commit/branch



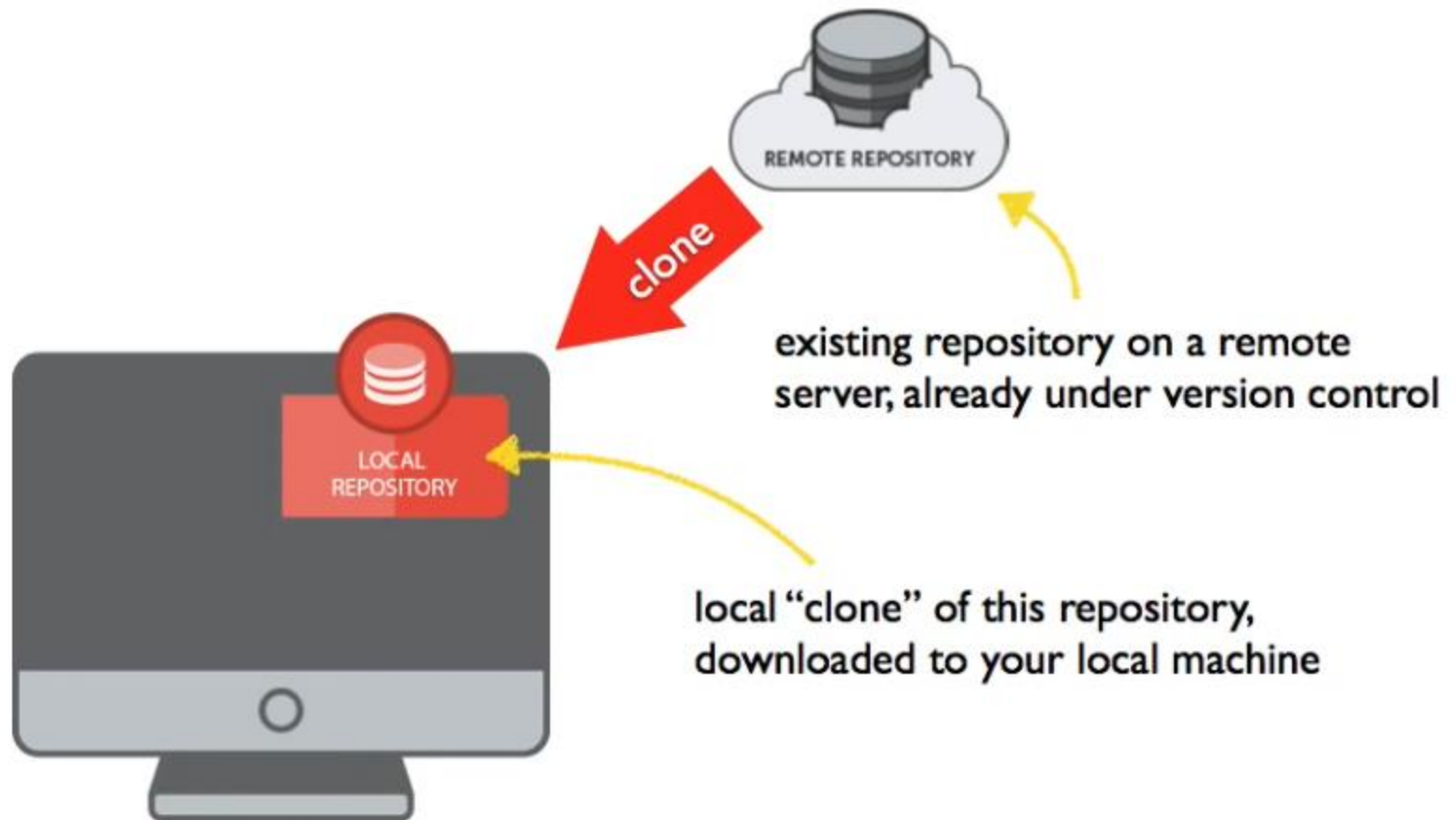
# git

local project...  
that is **not** under version control, yet

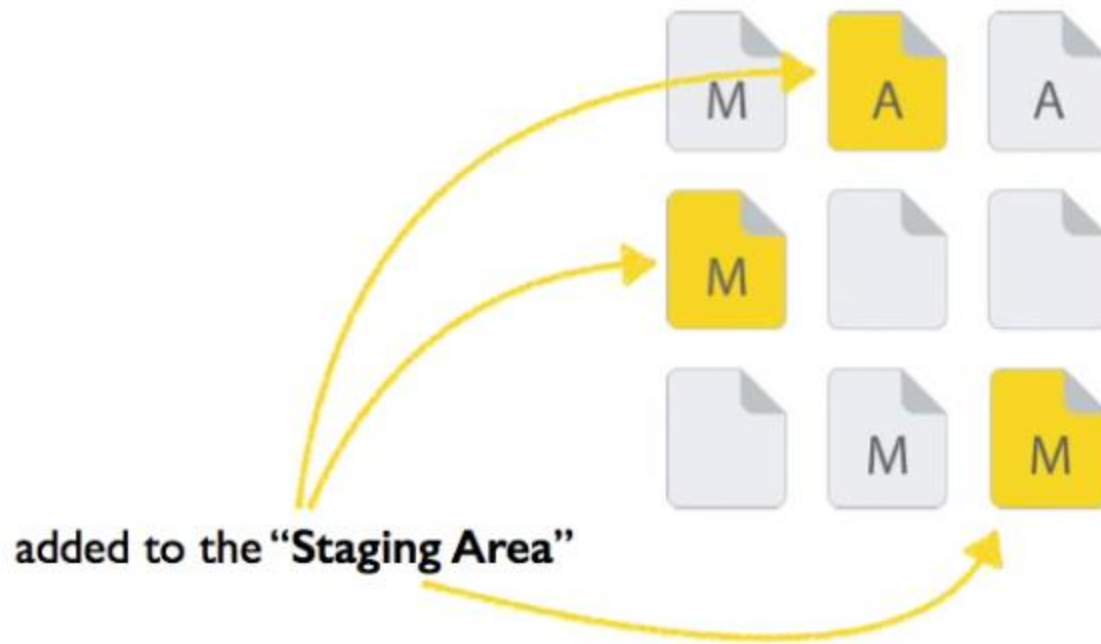








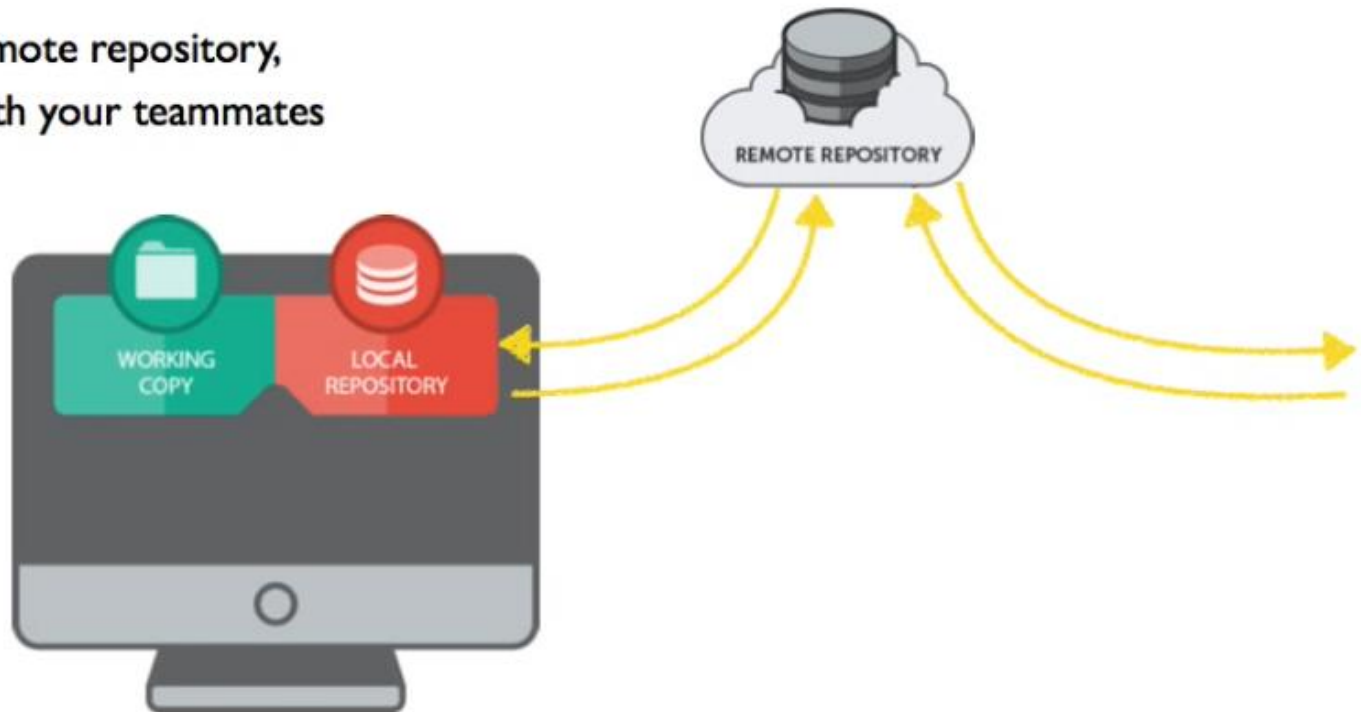
Which changes shall be part of the **next commit**?



Only **staged changes** are saved in the local repository as a new **commit**



**[Optional]** Through a remote repository,  
data can be exchanged with your teammates



- **Starting a new git repository**
  - Assume you have a file system containing your project's code modules
  - Navigate to that directory
  - **git init** creates your local repo

```
aparadise@APARADISE-LA MINGW64 ~  
$ mkdir MyProject  
  
aparadise@APARADISE-LA MINGW64 ~  
$ cd MyProject  
  
aparadise@APARADISE-LA MINGW64 ~/MyProject  
$ ls -l  
total 0  
  
aparadise@APARADISE-LA MINGW64 ~/MyProject  
$ ls -l  
total 4  
-rw-r--r-- 1 aparadise 1049089 2472 Jan 30 08:20 handle_form.php.txt  
  
aparadise@APARADISE-LA MINGW64 ~/MyProject  
$ git init  
Initialized empty Git repository in C:/Users/aparadise/MyProject/.git/  
  
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)  
$ ls -l  
total 4  
-rw-r--r-- 1 aparadise 1049089 2472 Jan 30 08:20 handle_form.php.txt  
  
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)  
$ ls -la  
total 16  
drwxr-xr-x 1 aparadise 1049089  0 Jan 31 08:04 ./  
drwxr-xr-x 1 aparadise 1049089  0 Jan 31 08:02 ../  
drwxr-xr-x 1 aparadise 1049089  0 Jan 31 08:04 .git/  
-rw-r--r-- 1 aparadise 1049089 2472 Jan 30 08:20 handle_form.php.txt
```

Now that I have a local repo, what can I do?

- Identify myself
- Check status

```
MINGW64:/c/Users/aparadise/MyProject

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ ls -la
total 16
drwxr-xr-x 1 aparadise 1049089  0 Jan 31 08:04 ./
drwxr-xr-x 1 aparadise 1049089  0 Jan 31 08:02 ../
drwxr-xr-x 1 aparadise 1049089  0 Jan 31 08:04 .git/
-rw-r--r-- 1 aparadise 1049089 2472 Jan 30 08:20 handle_form.php.txt

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git config --global user.name "Alan"

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git config --global user.email "alanparadise@gmail.com"

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
credential.helper=manager
user.name=Alan
user.email=alanparadise@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$
```



```
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        handle_form.php.txt

nothing added to commit but untracked files present (use "git add" to track)
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ :
```

## Working Copy

Your Project's Files



tracked

...modified



tracked

...unmodified



untracked

## Staging Area

Changes for Next Commit

## Local Repo

".git" Folder

```
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git add handle_form.php.txt

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   handle_form.php.txt

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$
```

## Working Copy

Your Project's Files



tracked  
...modified



tracked  
...unmodified



untracked

## Staging Area

Changes for Next Commit



staged

## Local Repo

".git" Folder



committed



```
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git commit -m "User Data Form Handler"
[master (root-commit) 196dec3] User Data Form Handler
1 file changed, 94 insertions(+)
create mode 100644 handle_form.php.txt

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ :
```

```
aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ git status
On branch master
nothing to commit, working tree clean

aparadise@APARADISE-LA MINGW64 ~/MyProject (master)
$ :
```

- **Git diff**
  - Compares two versions of a file
    - Modified versus last commit
    - “chunks”
    - Lines added, lines removed

Compared Files a/b

File Metadata

Markers for a/b

Chunk Header

Changes (1)

Chunk Header

Changes (2)

Chunk Header

Changes (3)

```
diff --git a/activesupport/test/test_cases.rb b/activesupport/test/test_cases.rb
index bbeb710..9b62295 100644
--- a/activesupport/test/constantize_test_cases.rb
+++ b/activesupport/test/constantize_test_cases.rb
@@ -34,6 +34,8 @@ module ConstantizeTestCases
   assert_equal Case::Dice, yield("Object::Case::Dice")
   assert_equal ConstantizeTestCases, yield("ConstantizeTestCases")
   assert_equal ConstantizeTestCases, yield("::ConstantizeTestCases")
+  assert_equal Object, yield("")
+  assert_equal Object, yield("::")
   assert_raises(NameError) { yield("UnknownClass") }
   assert_raises(NameError) { yield("UnknownClass::Ace") }
   assert_raises(NameError) { yield("UnknownClass::Ace::Base") }
@@ -43,8 +45,6 @@ module ConstantizeTestCases
   assert_raises(NameError) { yield("Ace::Base::ConstantizeTestCases") }
   assert_raises(NameError) { yield("Ace::Gas::Base") }
   assert_raises(NameError) { yield("Ace::Gas::ConstantizeTestCases") }
-  assert_raises(NameError) { yield("") }
-  assert_raises(NameError) { yield("::") }
end

def run_safe_constantize_tests_on
@@ -58,8 +58,8 @@ module ConstantizeTestCases
   assert_equal Case::Dice, yield("Object::Case::Dice")
   assert_equal ConstantizeTestCases, yield("ConstantizeTestCases")
   assert_equal ConstantizeTestCases, yield("::ConstantizeTestCases")
-  assert_nil yield("")
-  assert_nil yield("::")
+  assert_equal Object, yield("")
+  assert_equal Object, yield("::")
   assert_nil yield("UnknownClass")
   assert_nil yield("UnknownClass::Ace")
   assert_nil yield("UnknownClass::Ace::Base")
```

Chunk

Chunk

Chunk



```

aparadise@APARADISE-LA MINGW64 ~/myproject (master)
$ git diff handle_form.php
diff --git a/handle_form.php b/handle_form.php
index 7d600f6..4c8d40e 100644
--- a/handle_form.php
+++ b/handle_form.php
@@ -11,7 +11,7 @@
     </style>
</head>
<body width=60%>
-<?php # script 2.2 - handle_form.php
+<?php # script 2.2 - handle_form2.php

#shorthand for the form data
$name = $_REQUEST['name'];
@@ -19,7 +19,6 @@ $gender = $_REQUEST['gender'];
$team = $_REQUEST['team'];
$grade = $_REQUEST['grade'];

-var_dump ($_POST);

# Create the $name variable
if (!empty($_REQUEST['name'])) {
@@ -44,7 +43,7 @@ if (isset($_REQUEST['team'])) {
    $team = $_REQUEST['team'];

    if (in_array('football', $team) && (in_array('soccer', $team))) {
-    echo '<p class="error">Football & Soccer cannot be selected at the same time.';
+    echo '<p class="error">Foosball & Soccer cannot be selected at the same time.';
}

aparadise@APARADISE-LA MINGW64 ~/myproject (master)
$

```

- **Git log**
- **Git log --stat**

```
commit a3de79479a32c2dc50e126be7db2b1562790ce49
```

```
Author: Tobias Günther <tg@fournova.com>
```

```
Date: Wed Oct 1 14:23:29 2014 +0200
```

Restructure pages

```
commit 9d9b7bd259f9bd06b2ae77a90aae883921cbe4e1
```

```
Author: Tobias Günther <tg@fournova.com>
```

```
Date: Wed Sep 24 11:11:28 2014 +0200
```

Restructure index.html

```
commit 2b504bee4083a20e0ef1e037eea0bd913a4d56b6
```

```
Author: Tobias Günther <tg@fournova.com>
```

```
Date: Fri Jul 26 10:05:48 2013 +0200
```

Change headlines for about and imprint

```
commit 0023cdddf42d916bd7e3d0a279c1f36bfc8a051b
```

```
Author: Tobias Günther <tg@fournova.com>
```

```
Date: Fri Jul 26 10:04:16 2013 +0200
```

Add simple robots.txt

```
commit a3de79479a32c2dc50e126be7db2b1562790ce49
Author: Tobias Günther <tg@fournova.com>
Date:   Wed Oct 1 14:23:29 2014 +0200

    Restructure pages

error.html      | 43 -----
index.html      |  8 ++++----
new_page.html   |  6 ++++++
3 files changed, 10 insertions(+), 47 deletions(-)

commit 9d9b7bd259f9bd06b2ae77a90aae883921cbe4e1
Author: Tobias Günther <tg@fournova.com>
Date:   Wed Sep 24 11:11:28 2014 +0200

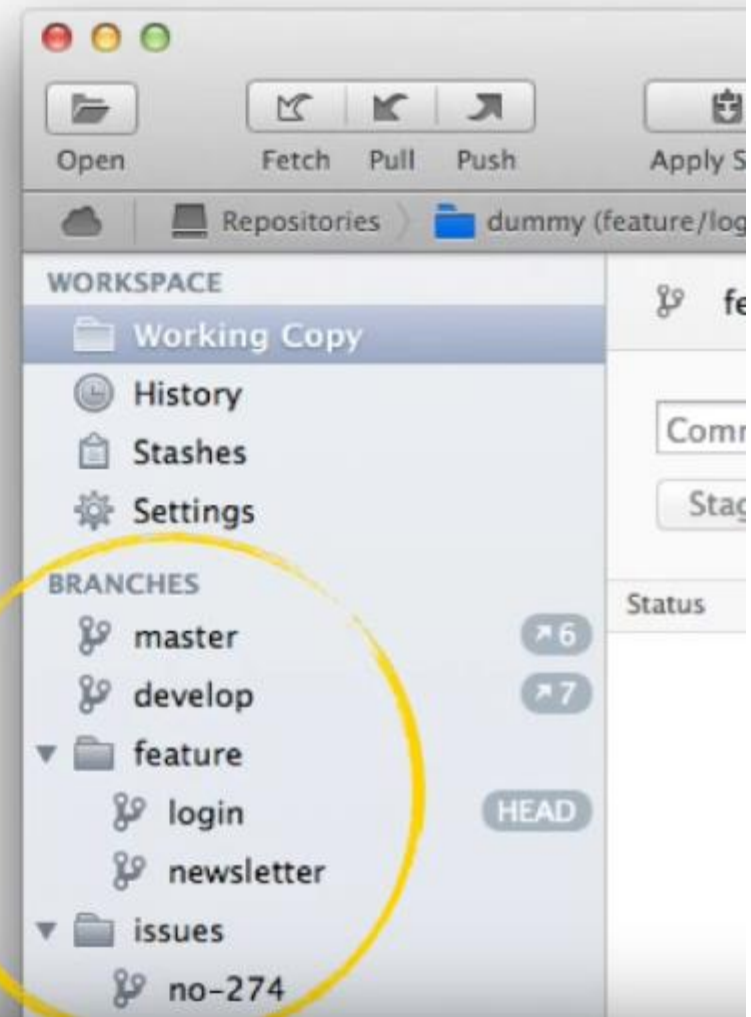
    Restructure index.html

index.html | 14 ++++++++----
1 file changed, 10 insertions(+), 4 deletions(-)

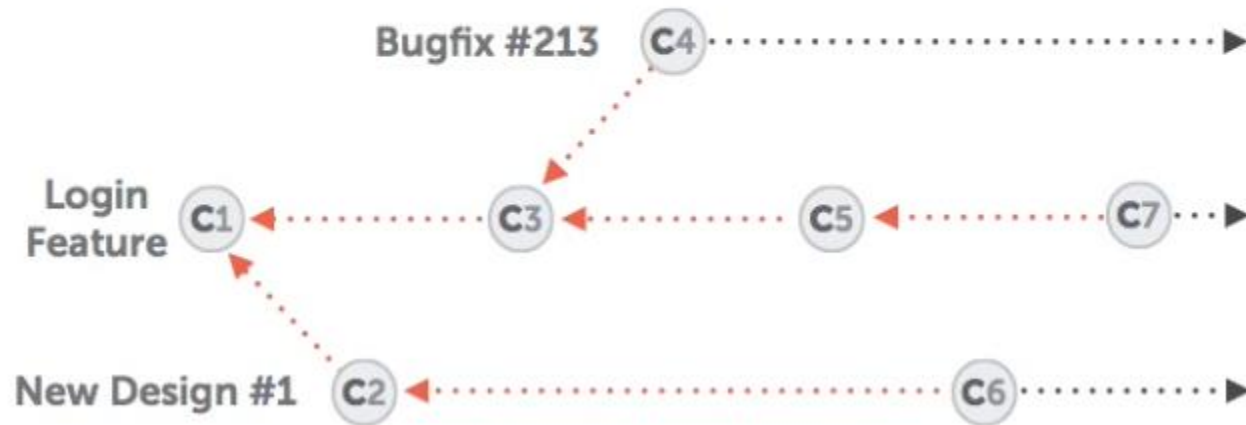
commit 2b504bee4083a20e0ef1e037eea0bd913a4d56b6
Author: Tobias Günther <tg@fournova.com>
Date:   Fri Jul 26 10:05:48 2013 +0200
```

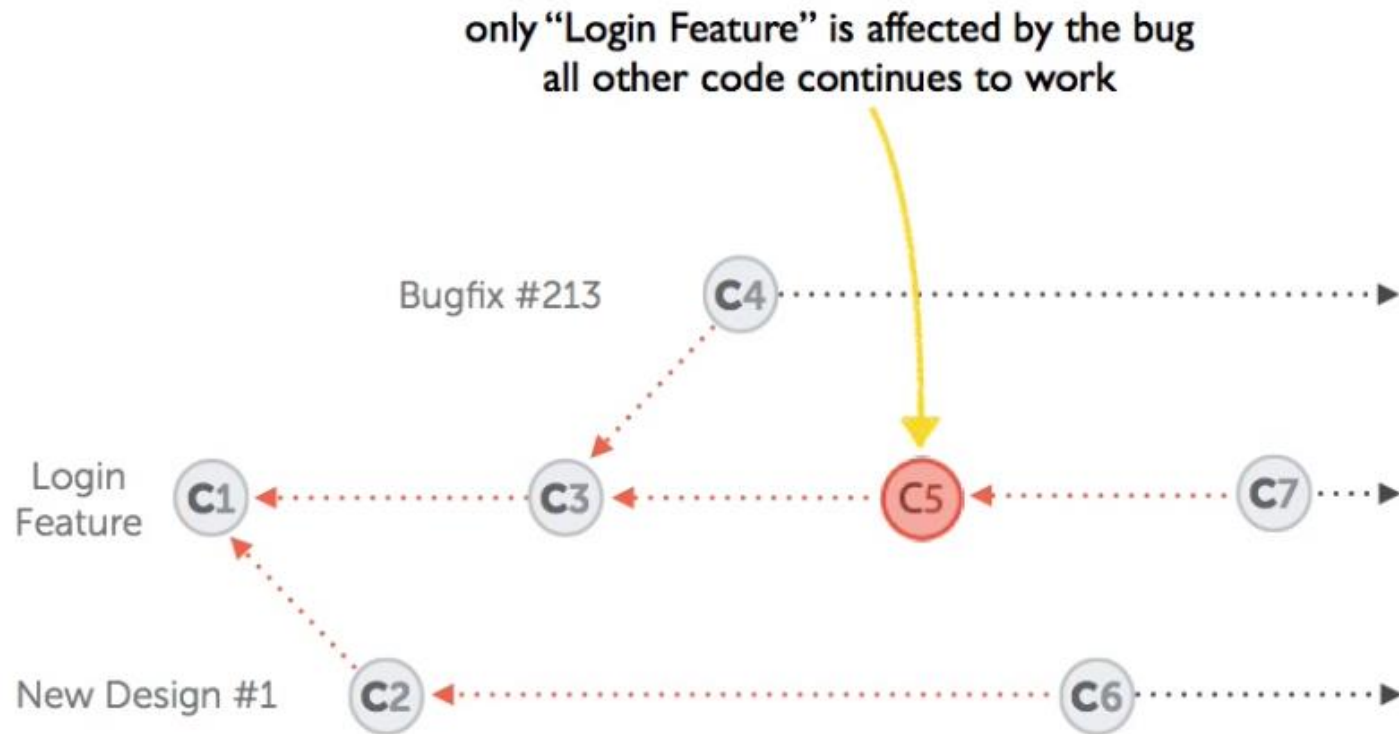
# *branches*

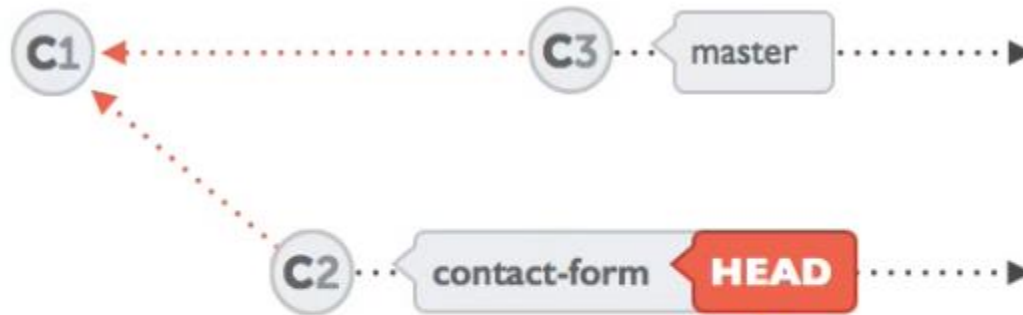
branches keep separate topics  
cleanly separated from each other



**Work safely in separate contexts.**

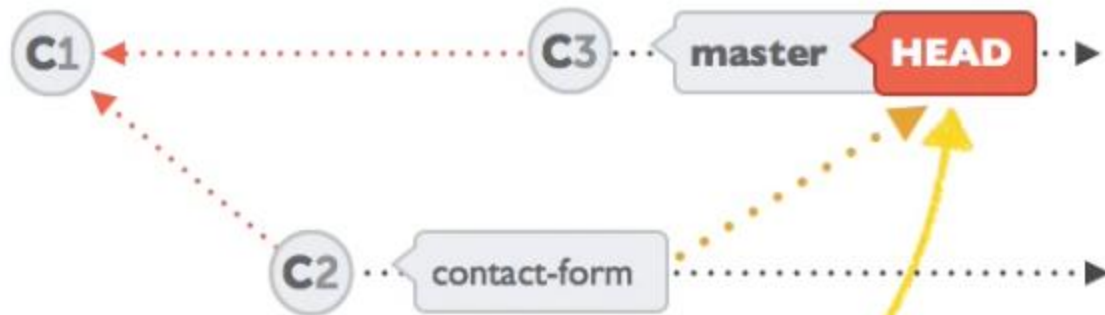




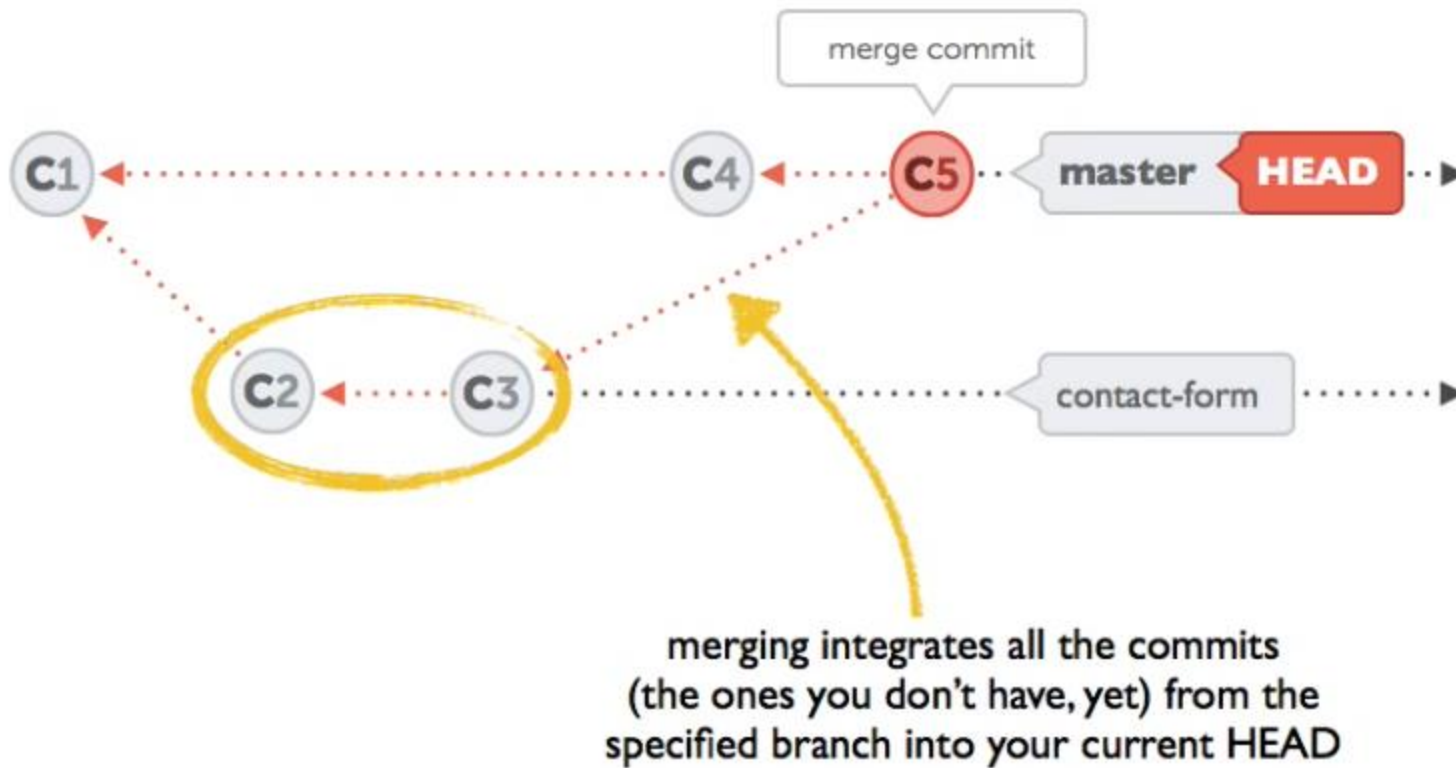


at any time, only **one** branch is  
active = checked out = HEAD





the “**checkout**” command moves the HEAD pointer to a different branch - and thereby makes that branch active



- **Links to online video training**
- **<https://try.github.io>**  
**(start here)**
- **<https://www.git-tower.com/learn/git/videos>**  
**(11 free videos)**

- **Recap of most important git tasks**
  - git help
  - git init – prepares your local git
  - git add – stages for commit
  - git clone – copies central repo to local copy
  - git commit – copies staged work to repo
  - git branch – shows all branches
  - git checkout – switches branches
  - git merge – merges with master
  - git diff – shows changes
  - git fetch – moves code from central repo to local
  - git pull – fetch + merge to local

**For your group project, you must create three git repositories, and share them with the instructor and all TAs**

1. A record of all team meeting notes (one file per meeting)
2. A repo for all Milestones submitted for the project (one file for each milestone)
3. A repo for all project files (code, etc.)