

Agenda

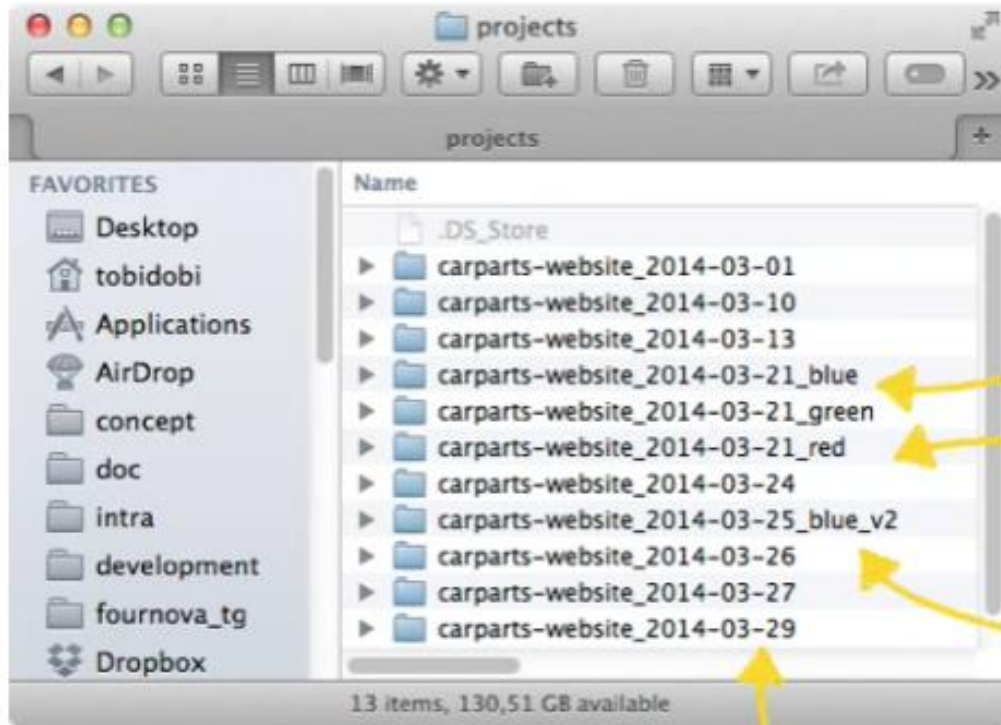
1. The need for Version Control
2. Using git for version control

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.¹ And for every developer added to the team, the overhead gets worse.²

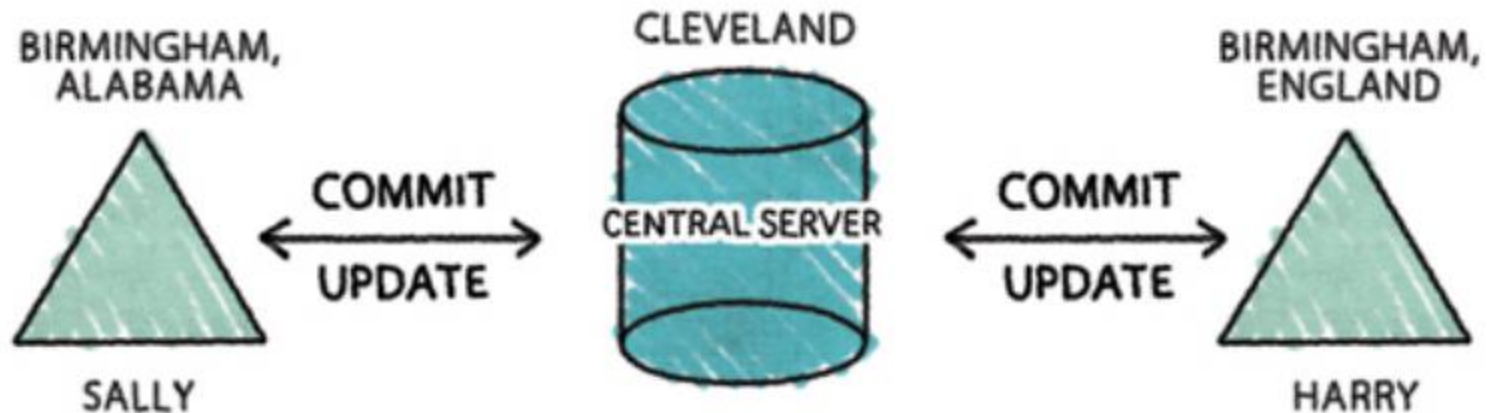
² The function is $n*(n-1)/2$.

I want a REPOSITORY:

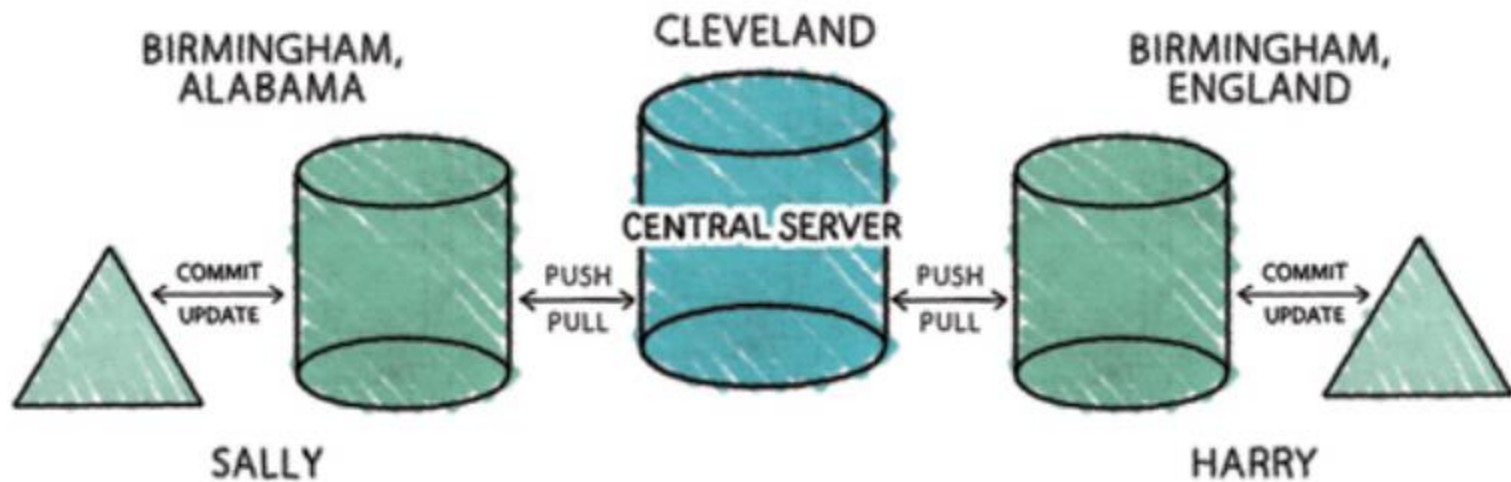
- Coordinates across multiple developers on a team
- Keeps History
- Allows me to revert back to a prior version of my code.
- Allows me and others to work on the same module at the same time.
- Tracks WHO is working on WHAT
- Prevents us from clobbering each others' updates.
- Describes each version 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**

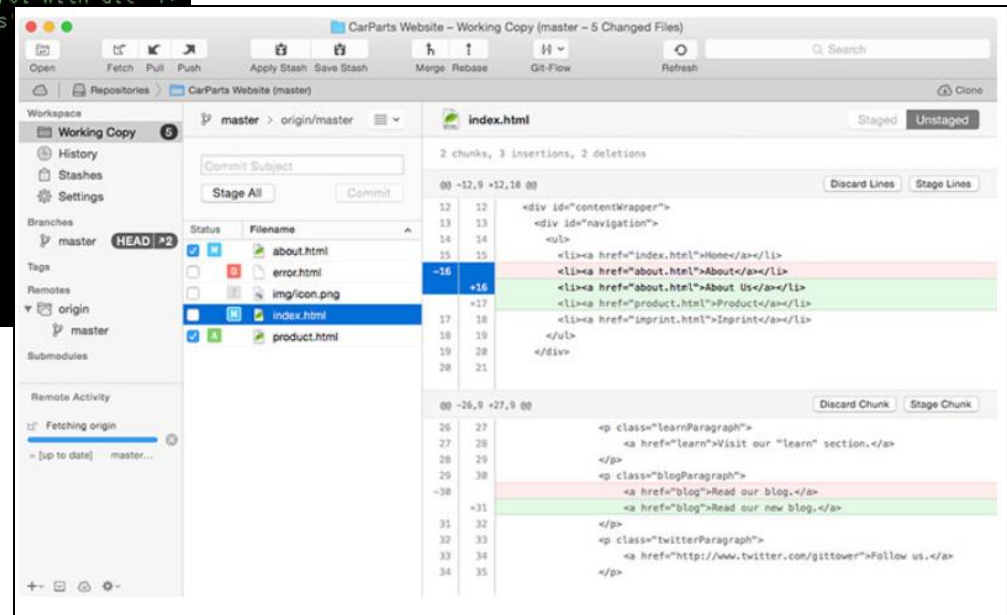
Version Control

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](#))



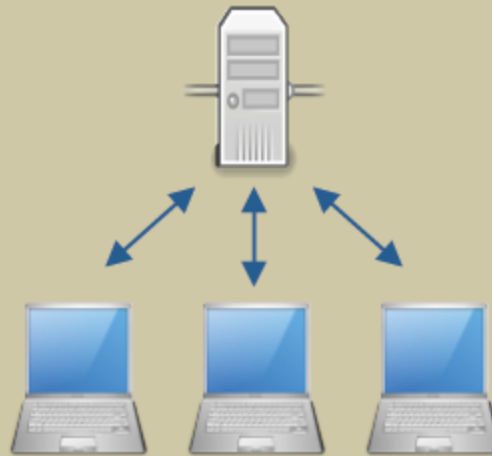
2000: Subversion ([SVN](#)), [BitKeeper](#)



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

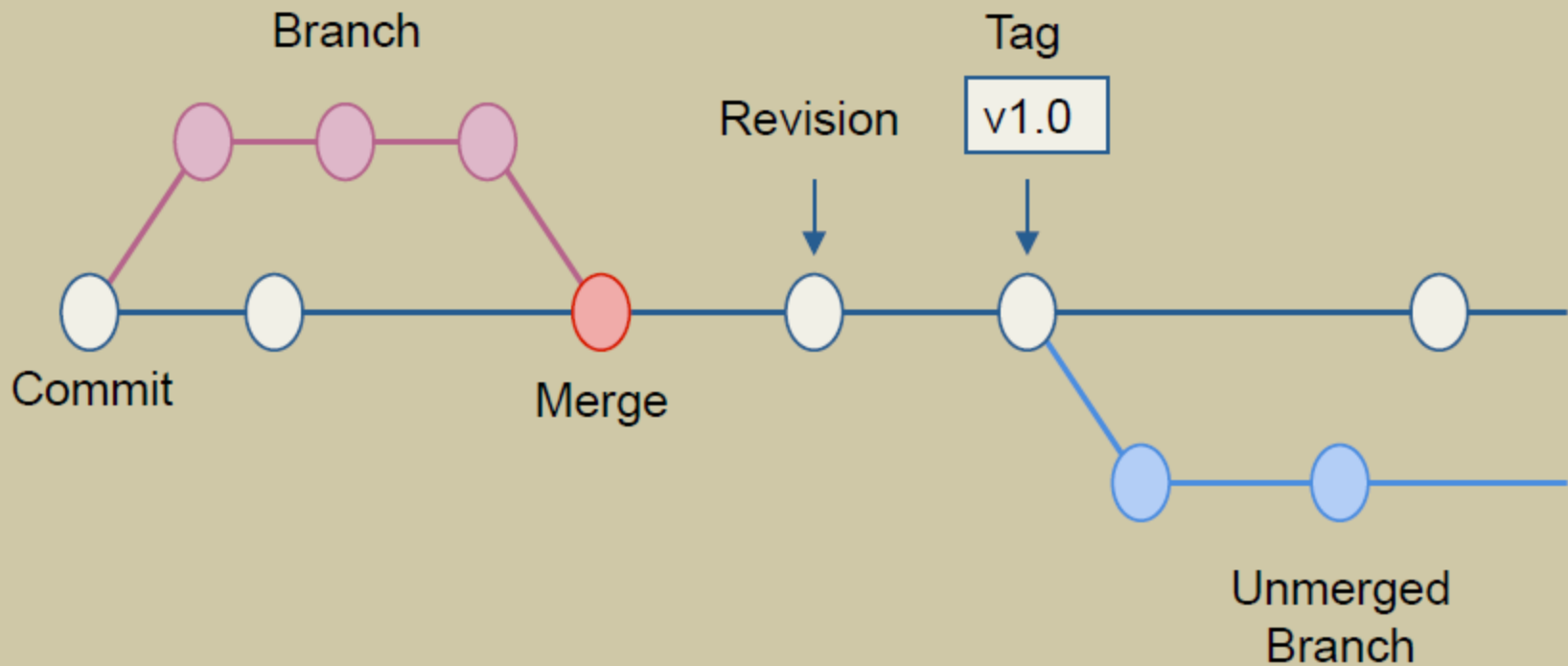
Branch-Based



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



Code Control Methods

- **LOCO** : **L**ock **O**n **C**heck **O**ut

- Developer 1 “checks out” a file for editing.
- Repository puts a lock on the file.
- Other developers can’t “check out” file until developer 1 performs a “check in”.



Code Control Methods

- **MOM** : **M**erge **O**n **M**odify



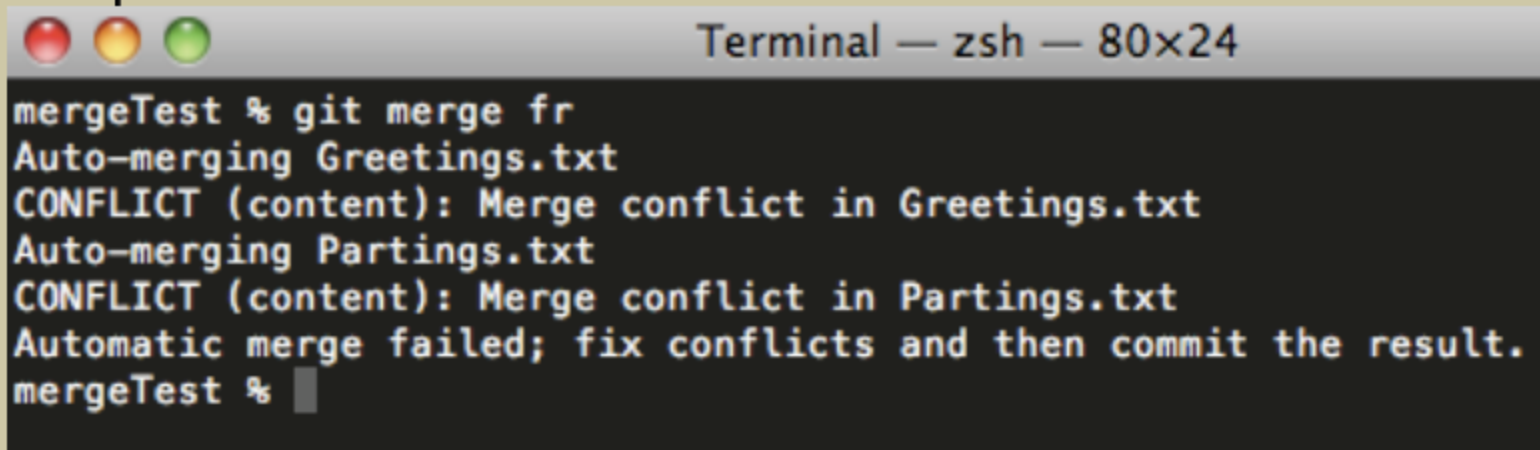
- Developer 1 gets the latest version from the repository.
 - *No locks places on any files!!!*
 - Other developers may also edit the file.
- Changes committed back to repository; merged with current repository file.

Method Comparison

LOCO	MOM
Locks files for editing	Does not lock files for editing
No concurrent file access	Allows concurrent file access
Check-in replaces current version	Changes merged with current version
No conflict on check-in	Possible conflict when committing
Works for any file type	Works best on text-based files.
Central repository model	Central or decentralized repository supported

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 shows the TortoiseMerge application with three panes. The top-left pane, titled 'Theirs - HelloClass.cs.r16', contains the following code:

```

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.";
18 -----return theGreeting;
19 -----}
20 }

```

The top-right pane, titled 'Mine - HelloClass.cs.mine', contains the following code:

```

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

```

The bottom pane, titled 'Merged - HelloClass.cs', shows the result of the merge. It contains the following code:

```

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 }

```

The status bar at the bottom indicates 'Left View: UTF8 BOM CRLF / - 1 / + 4', 'Right View: UTF8 BOM CRLF / - 1 / + 1', 'Conflicts: 2', 'CAP', 'NUM', and 'SCRL'.

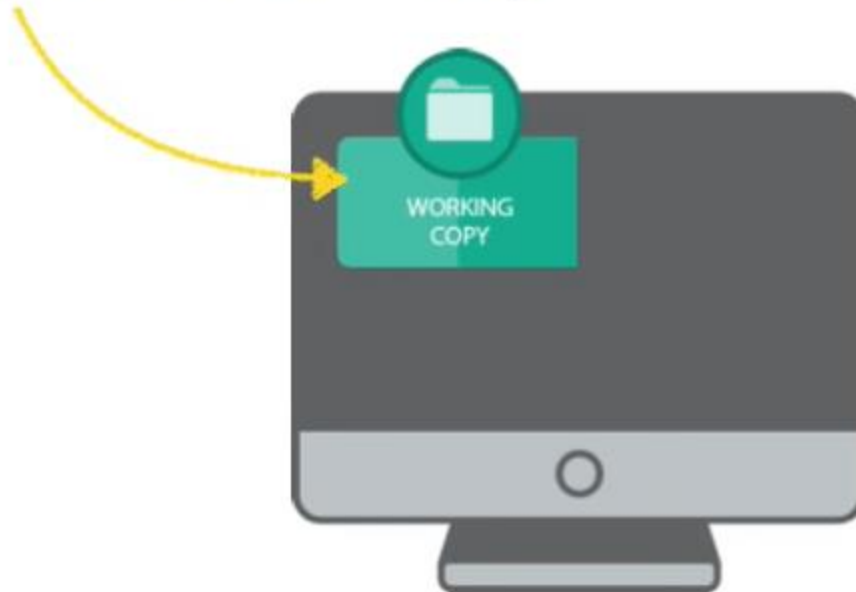
Best Practices

- 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

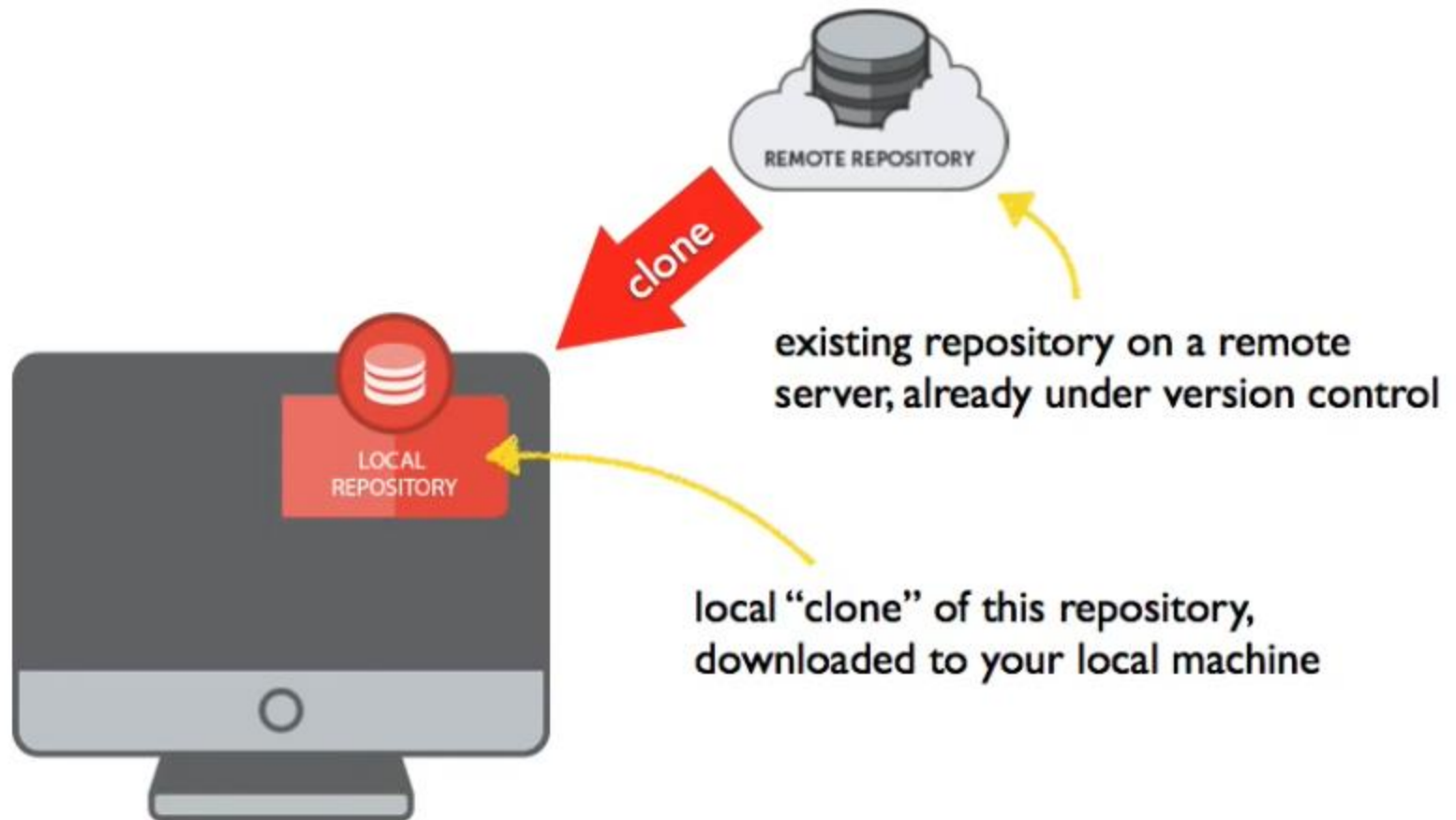


git

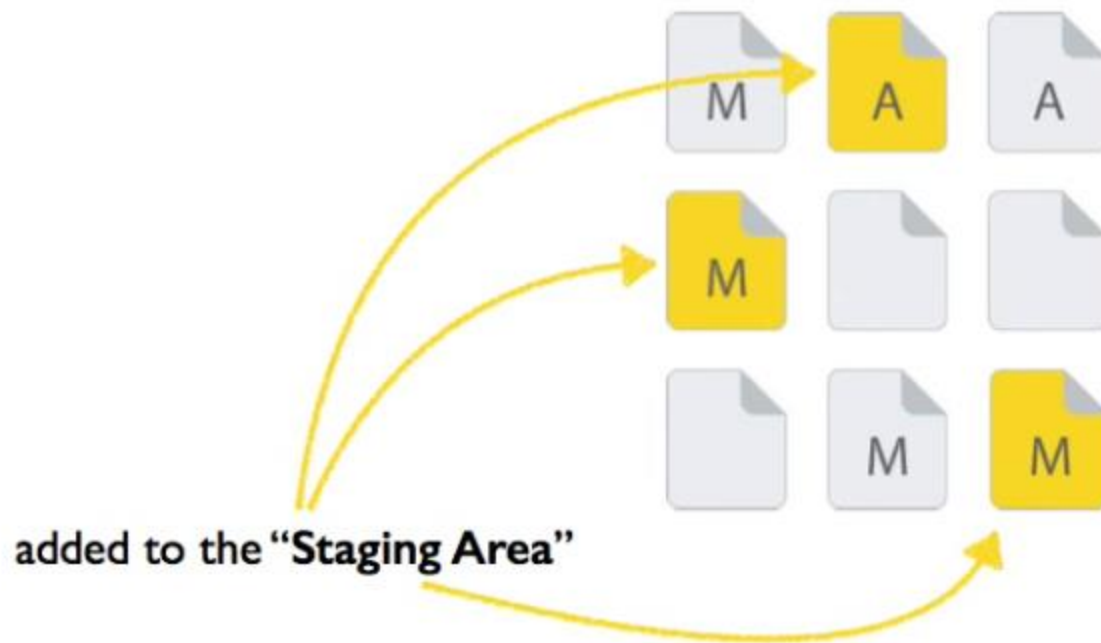
local project...
that is **not** under version control, yet







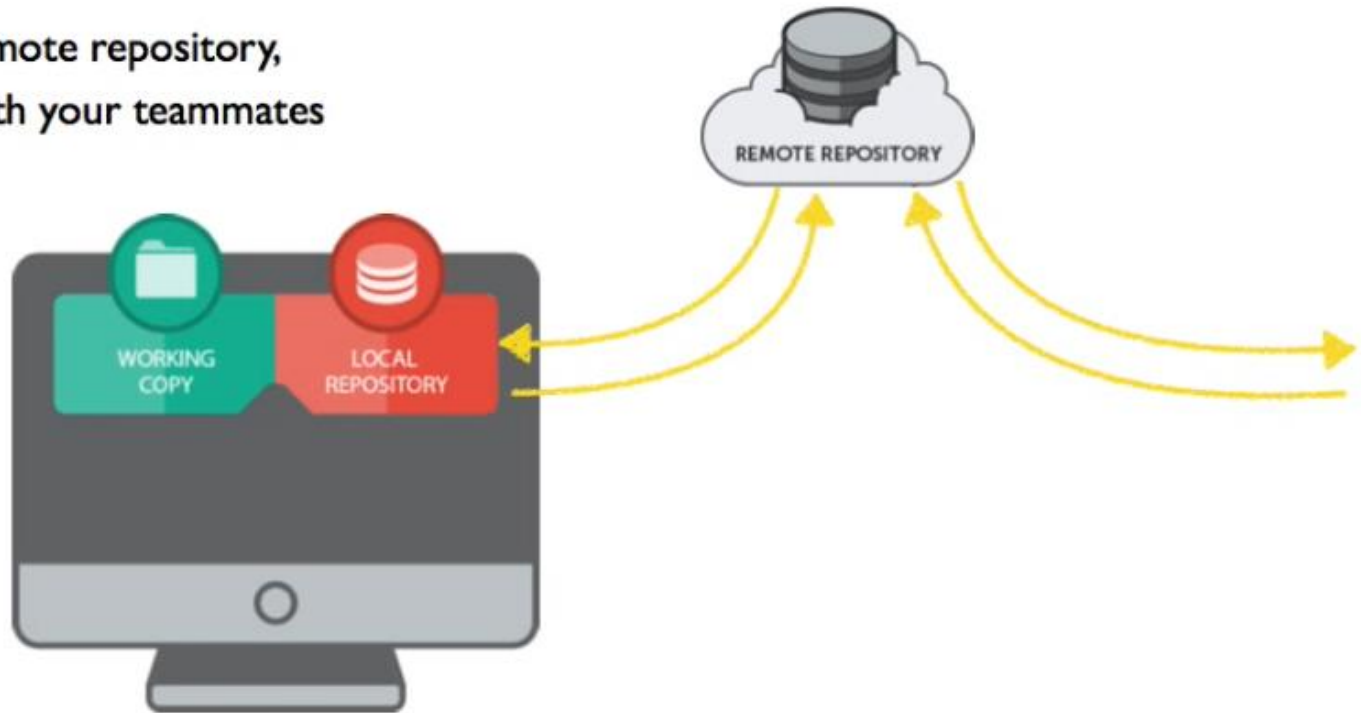
Which changes shall be part of the **next commit**?



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



```
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>
```

```
@@ -29,8 +29,8 @@
```

```
<p class="blogParagraph">
```

```
<a href="http://www.git-tower.com/blog">Read our blog</a> for interesting articles
```

```
</p>
```

```
- <p class="twitterParagraph">
```

```
- <a href="http://www.twitter.com/gittower">Follow us</a> on Twitter.
```

```
+ <p class="followTwitterParagraph">
```

```
+ <a href="http://www.twitter.com/gittower">Follow us</a> on our Twitter account.
```

```
</p>
```

```
<br />
```

- **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

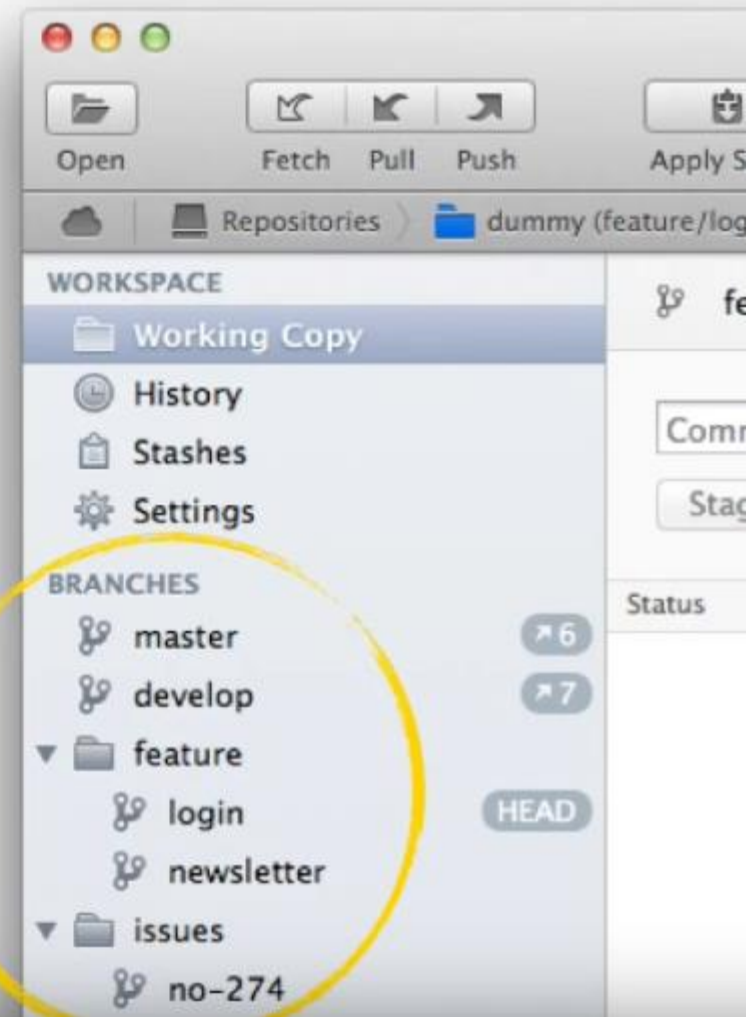
    Restructure index.html

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

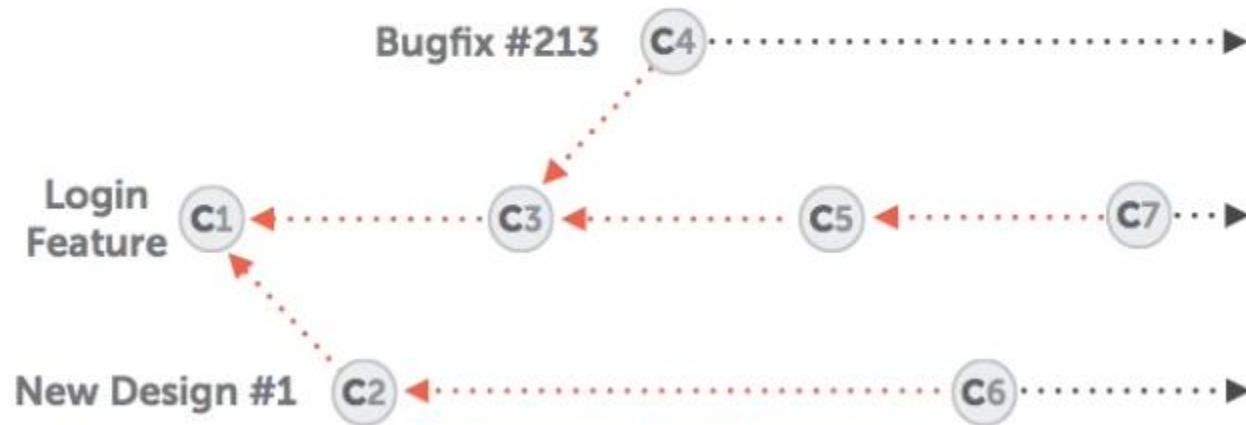
commit 2b504bee4083a20e0ef1e037eea0bd913a4d56b6
Author: Tobias Günther <tg@fournova.com>
Date:   Fri Jul 26 10:05:48 2013 +0200
```

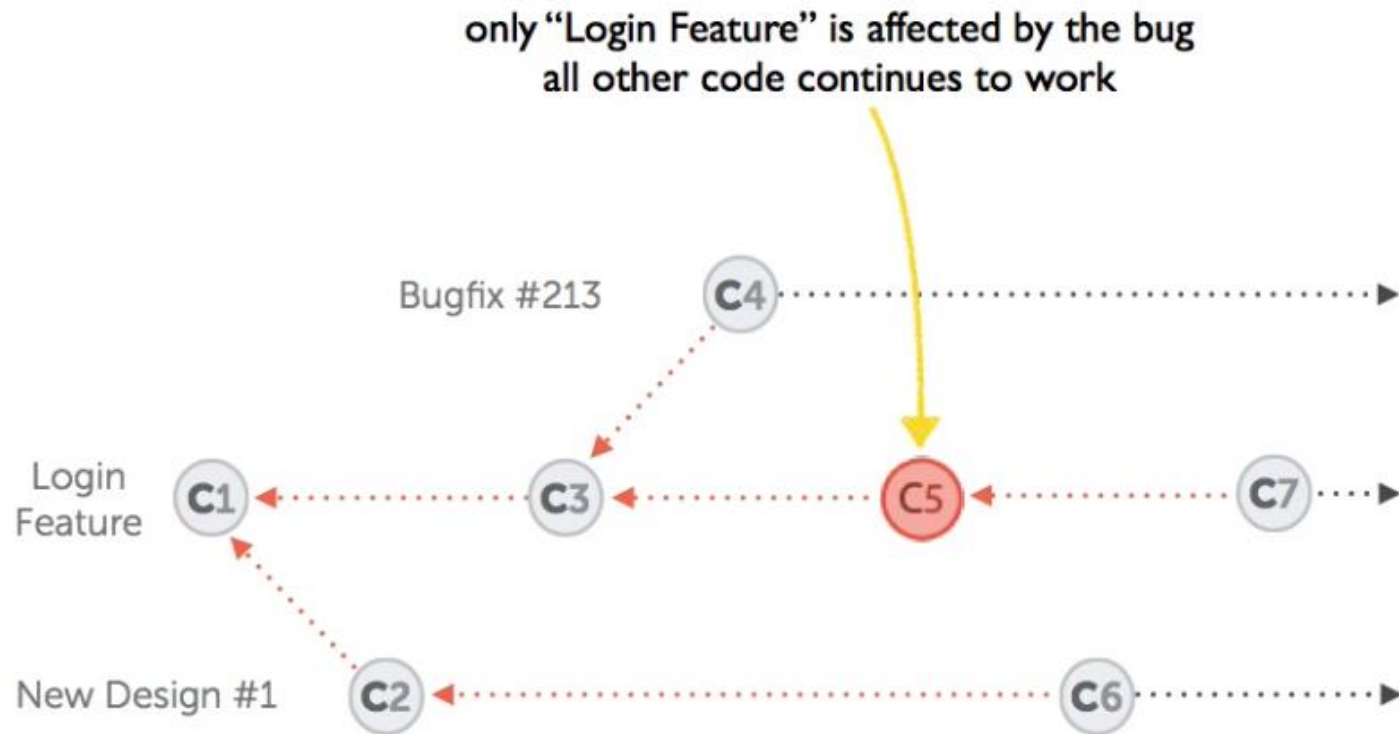
branches

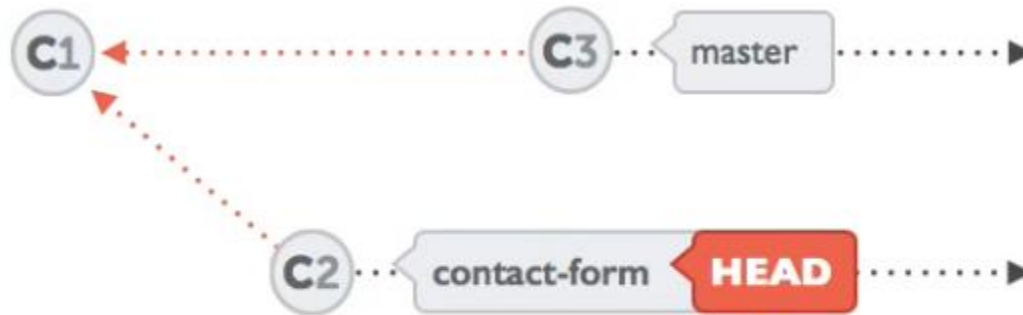
branches keep separate topics
cleanly separated from each other



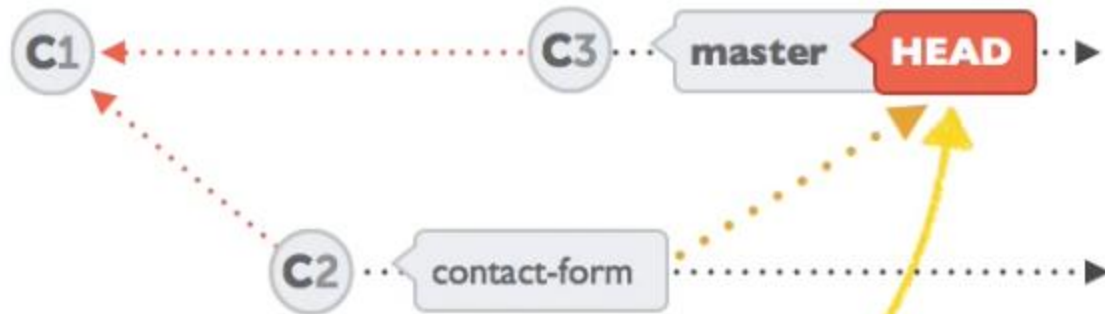
Work safely in separate contexts.



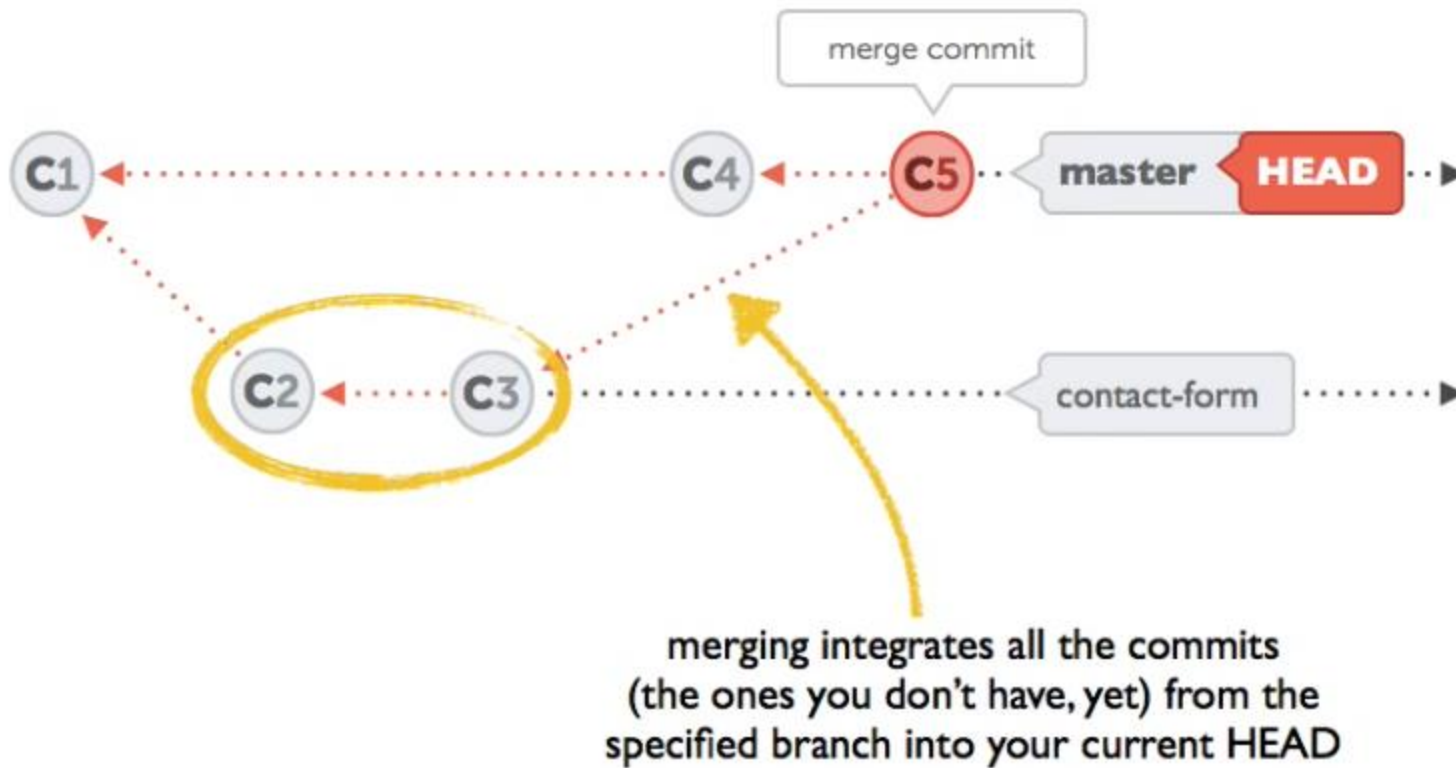




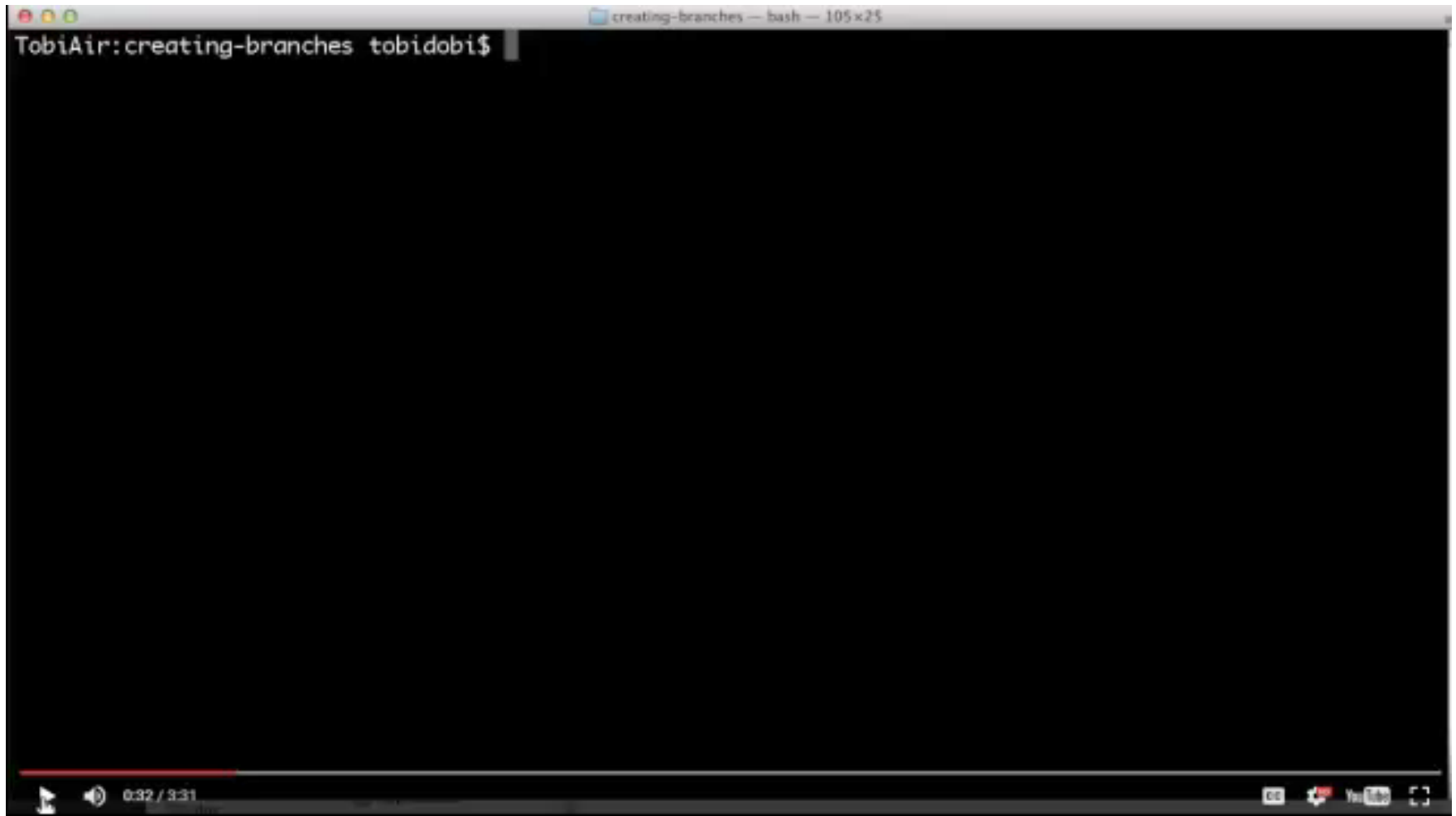
at any time, only **one** branch is
active = checked out = **HEAD**



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



git branch



- **Links to online video training**
- **<https://try.github.io/levels/1>**
(start here)
- **<https://www.git-tower.com/learn/git/videos>**
(11 free videos)