VERSION CONTROL

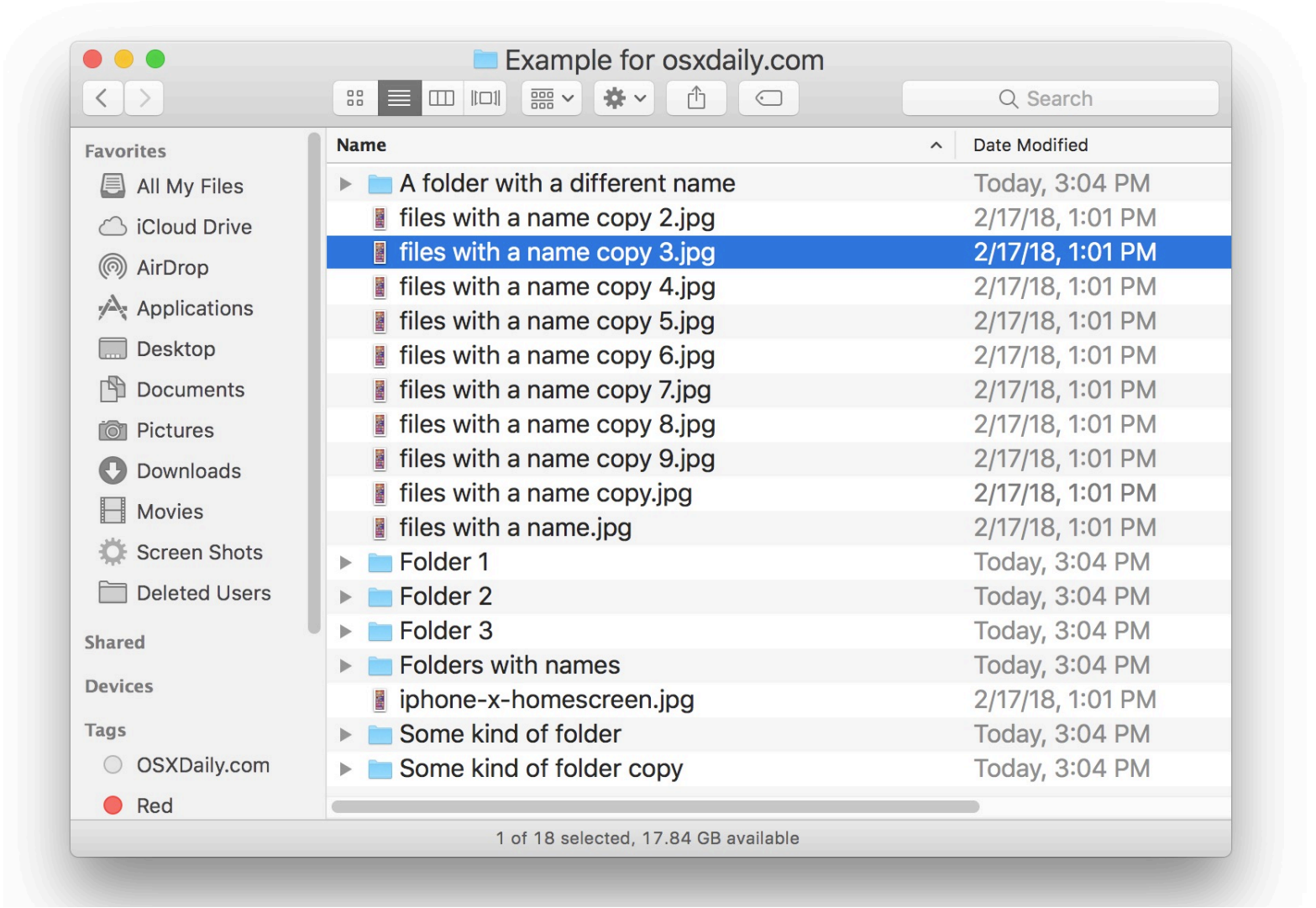
WEEK 4 – 09/18/2019

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

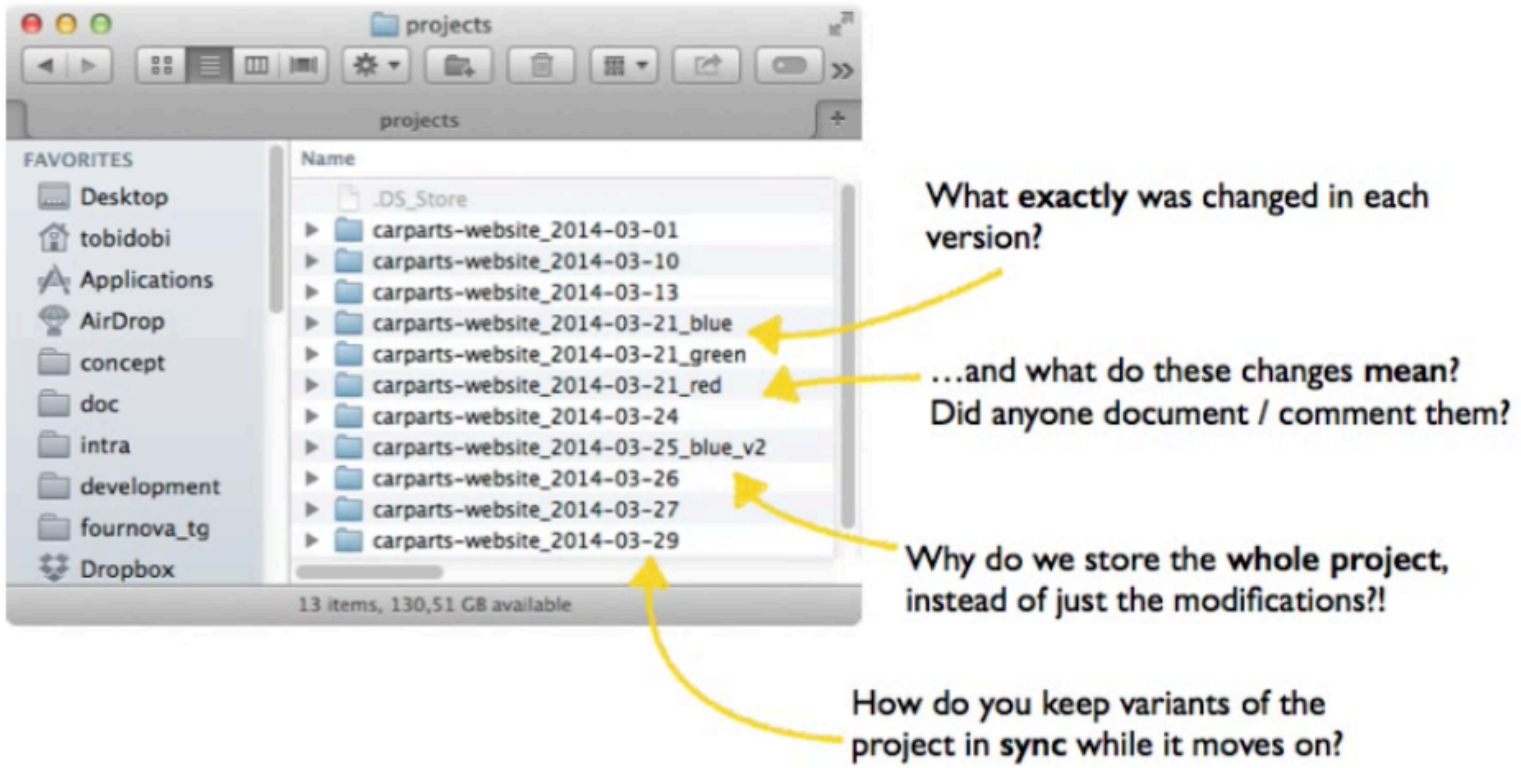
Where do I store my code?

EXAMPLE



shorturl.at/clmqZ

EXAMPLE


projects
projects
FAVORITES
Desktop
tobidobi
Applications
AirDrop
concept
doc
intra
development
fournova_tg
Dropbox
Name
.DS_Store
carparts-website_2014-03-01
carparts-website_2014-03-10
carparts-website_2014-03-13
carparts-website_2014-03-21_blue
carparts-website_2014-03-21_green
carparts-website_2014-03-21_red
carparts-website_2014-03-24
carparts-website_2014-03-25_blue_v2
carparts-website_2014-03-26
carparts-website_2014-03-27
carparts-website_2014-03-29
13 items, 130,51 GB available
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

I WANT A REPOSITORY

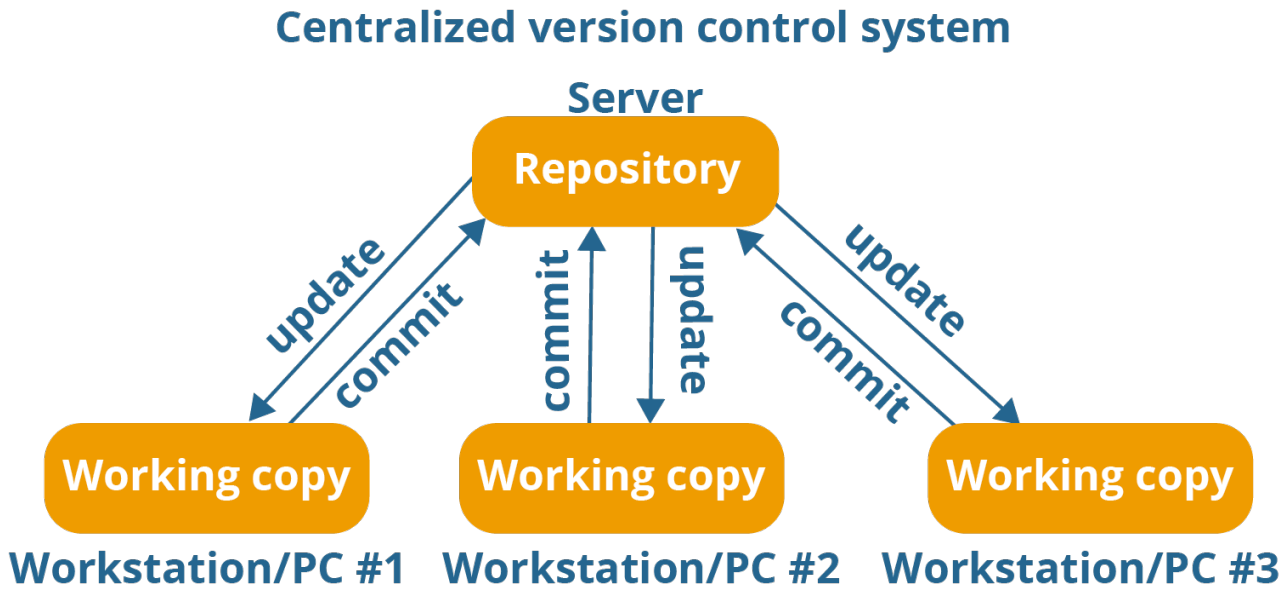
- Coordinates across multiple developers on a team
- Keeps History
- Allows me to revert back to a prior version of my code.
- Allows me and others to work on the same module at the same time.
- Tracks WHO is working on WHAT
- Prevents us from clobbering each others' updates.
- Describes each version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

TWO DESIGNS FOR VERSION CONTROL SYSTEMS

CENTRAL REPOSITORY "VCS"

DISTRIBUTED REPOSITORY "DVCS"

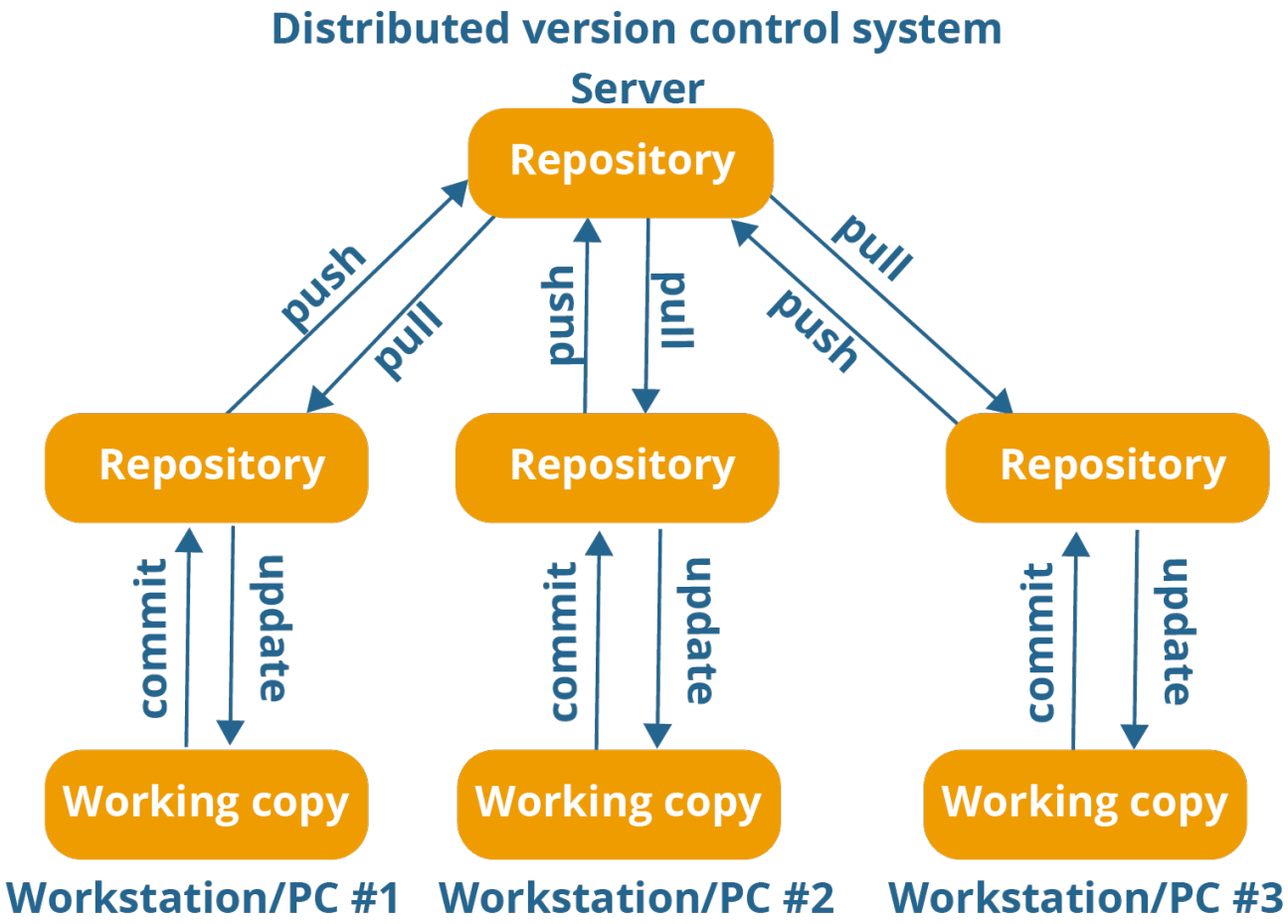
CENTRAL REPOSITORY "VCS"



POPULAR OPEN SOURCE VCS TOOLS: (CENTRAL REPO)

- Concurrent Versions System (CVS)
- CVSNT
- OpenCVS
- Subversion
  - https://subversion.apache.org/features.html

DISTRIBUTED REPOSITORY "VCS"

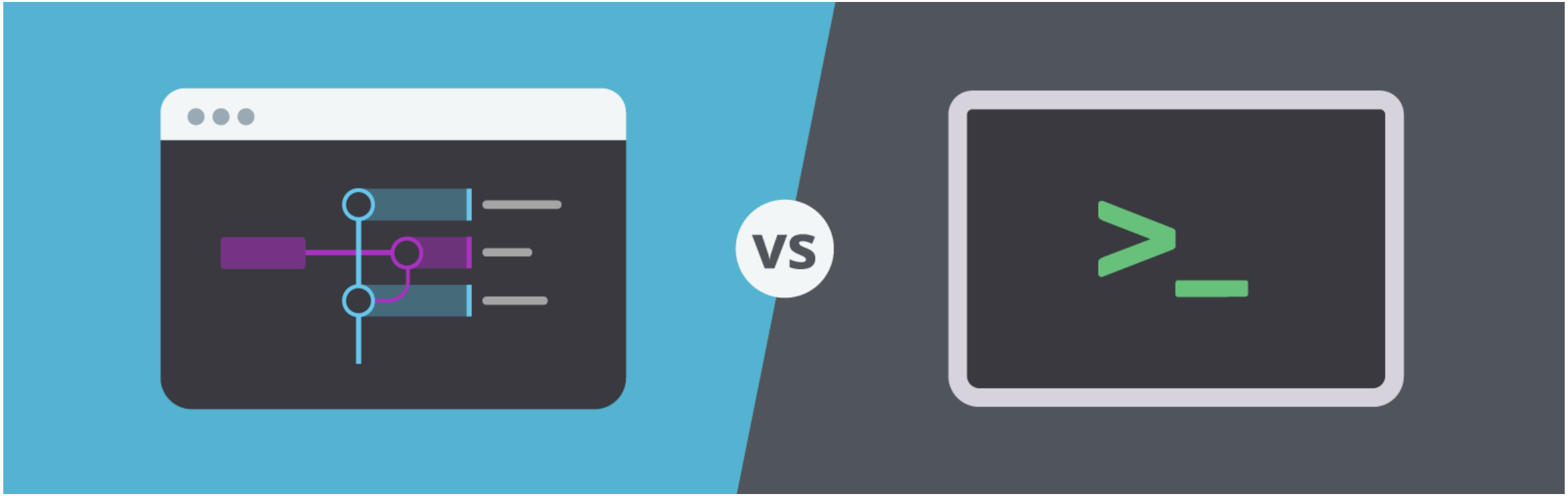


WHY DVCS?

- 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

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



OPTIONS ➡ GUI VS COMMAND LINE

POPULAR FREE GUI TOOLS (FOR GIT)

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

THREE GENERATIONS OF VERSION CONTROL

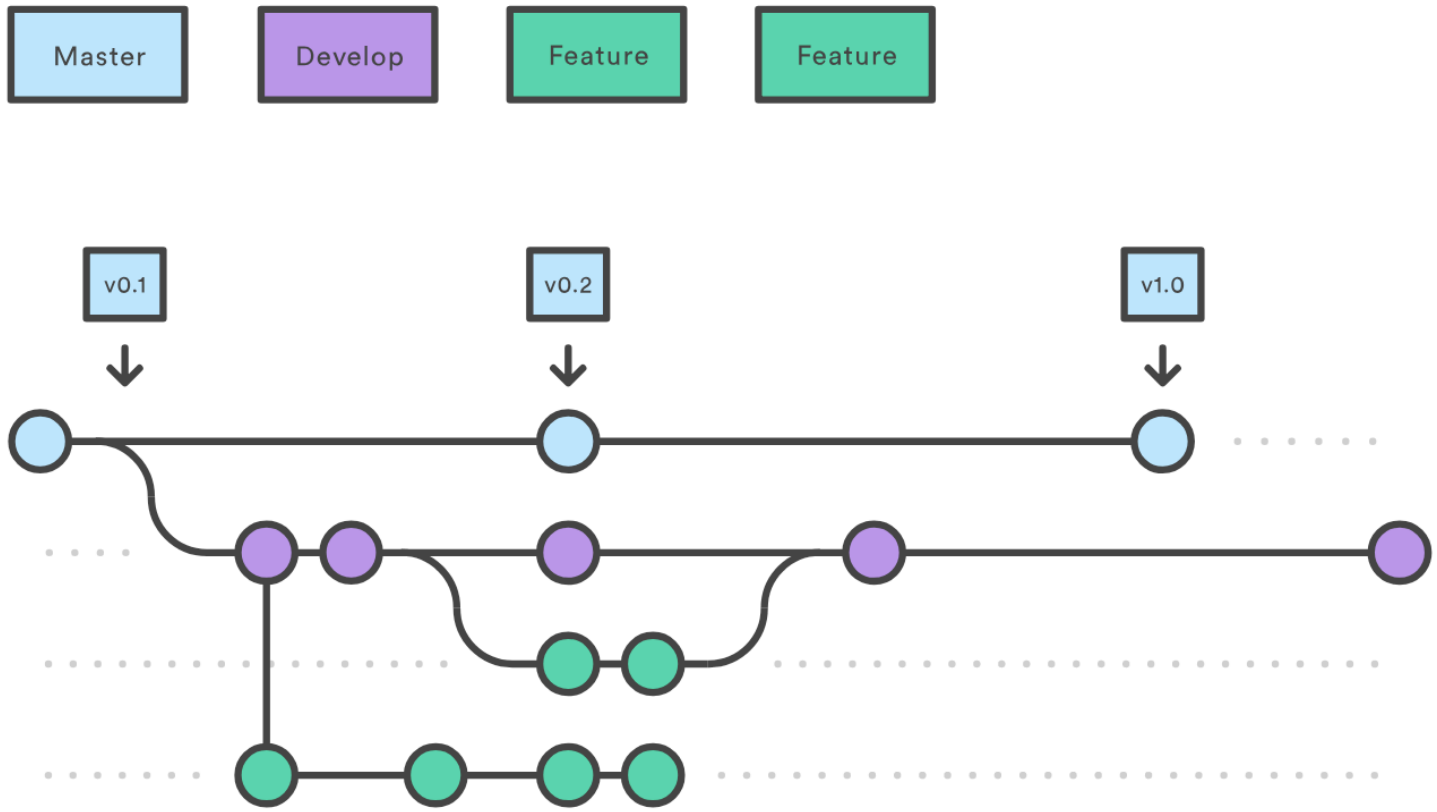
Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	CVS, SourceSafe, Subversion, Team Foundation Server
Third	Distributed	Changesets	Commit before merge	Bazaar, Git, Mercurial

shorturl.at/bqxCJ

TERMINOLOGY

- **Repository**: Version controlled collection of files/code
- **Commit**: Write/Save changes to local repository
- **Revision**: A specific version of repo/file
- **Tag**: A logically named/labeled version
- **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



BEST PRACTICES

DO

Commit Early, Commit Often

Use clear commit messages

Communicate when resolving merge conflicts

Only commit related changes in one commit

One topic per branch

Only commit source code

DON'T

Commit compiled files

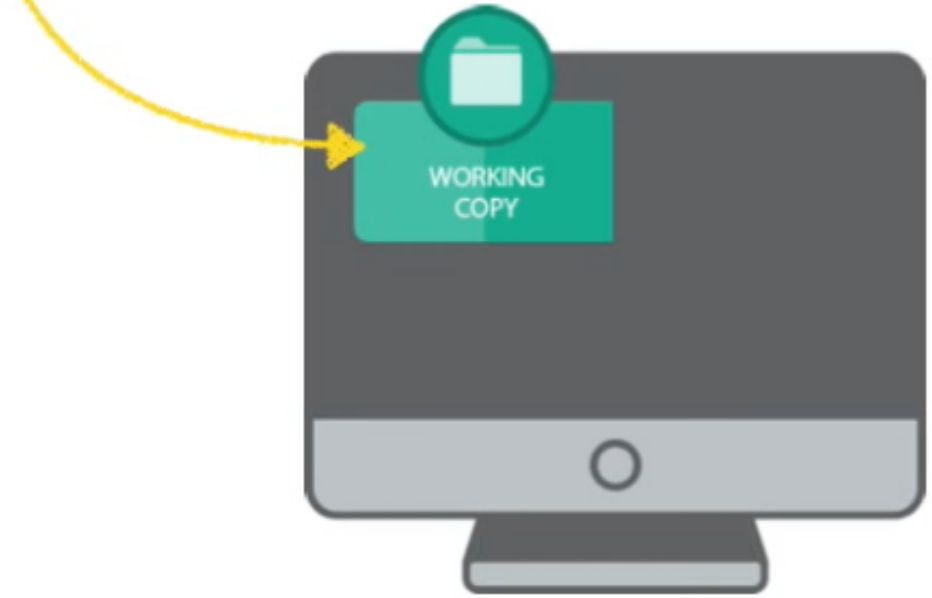
Commit secure info (eg: credentials, passwords)

Merge broke code (don't break the master branch)

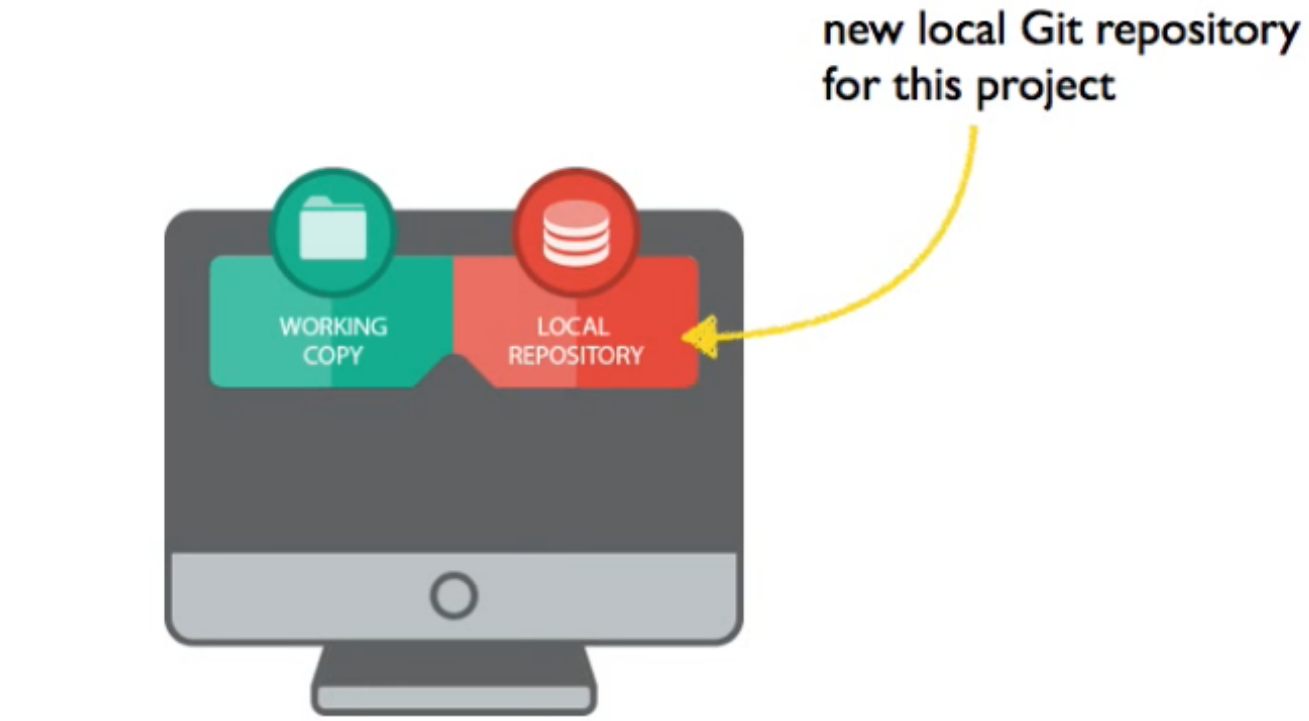
Combine multiple topics per commit/branch

GIT

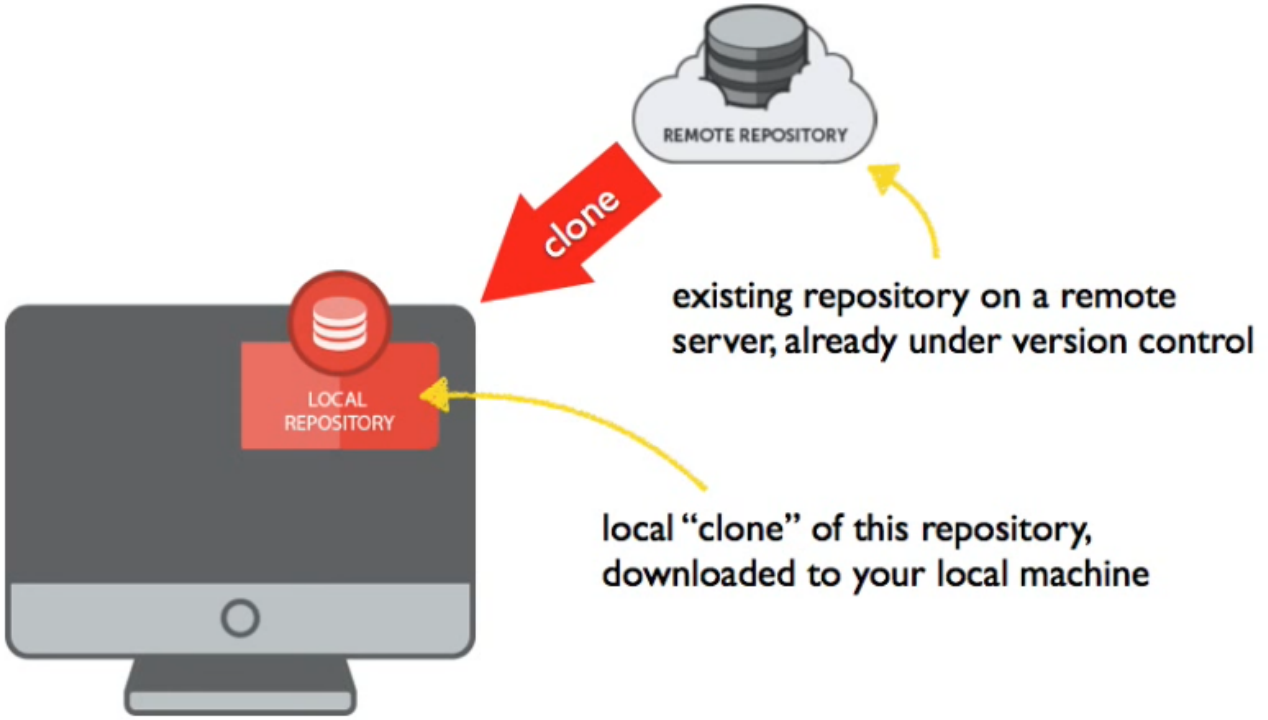
local project…
that is **not** under version control, yet



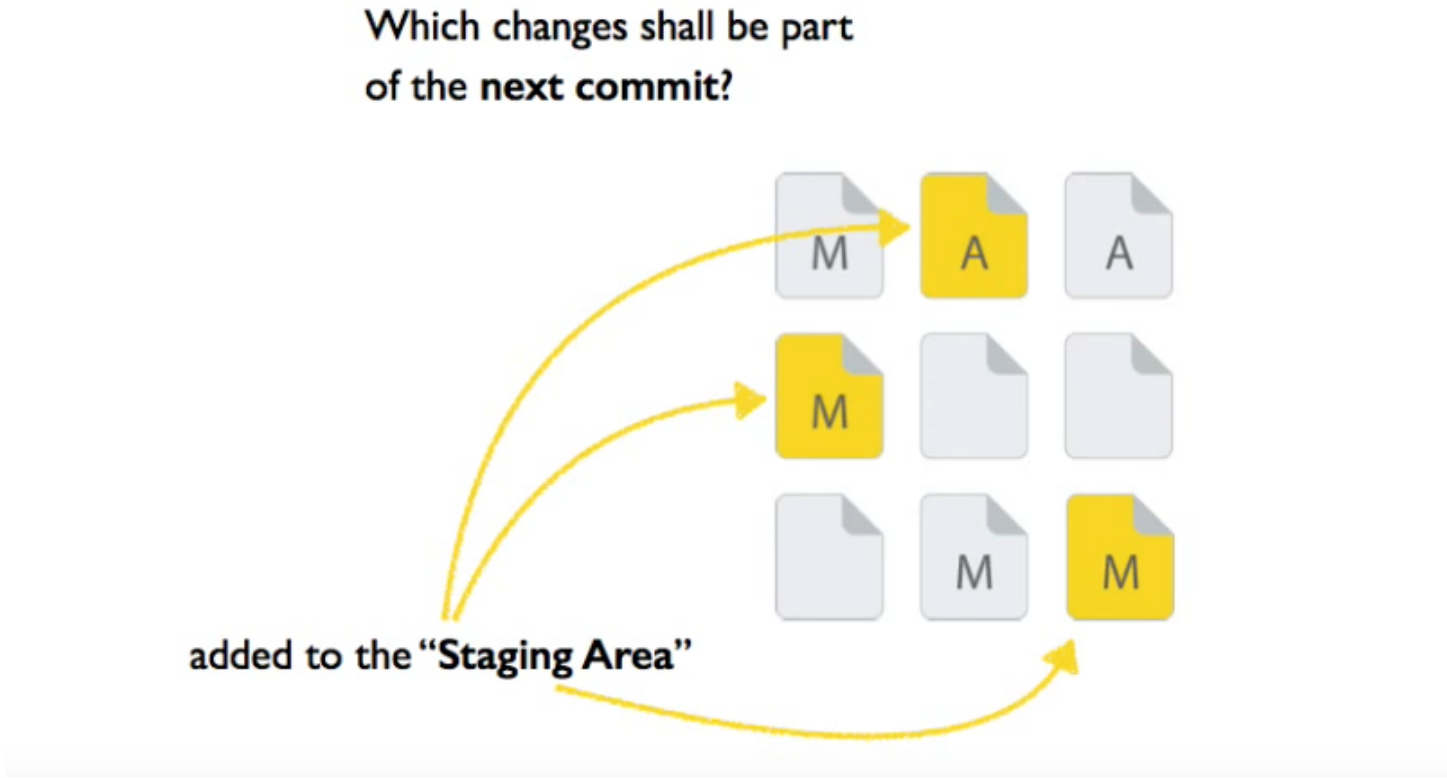
GIT


new local Git repository
for this project
WORKING
COPY
LOCAL
REPOSITORY

GIT


REMOTE REPOSITORY
clone
existing repository on a remote server, already under version control
LOCAL REPOSITORY
local "clone" of this repository, downloaded to your local machine

GIT

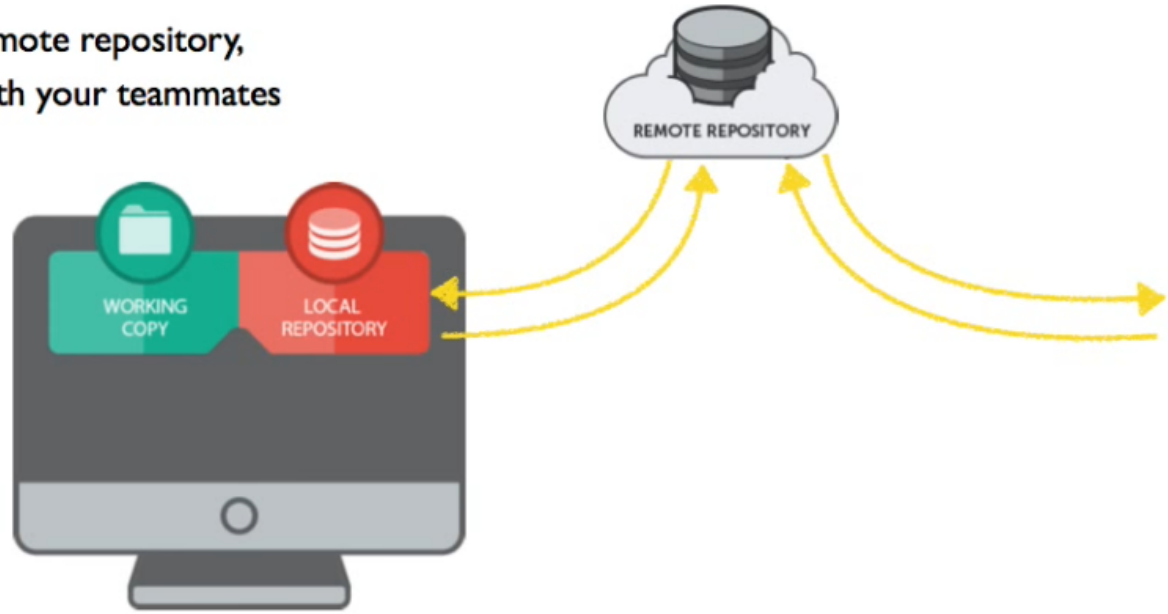

Which changes shall be part
of the next commit?
M
A
A
M
M
M
added to the "Staging Area"

GIT

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

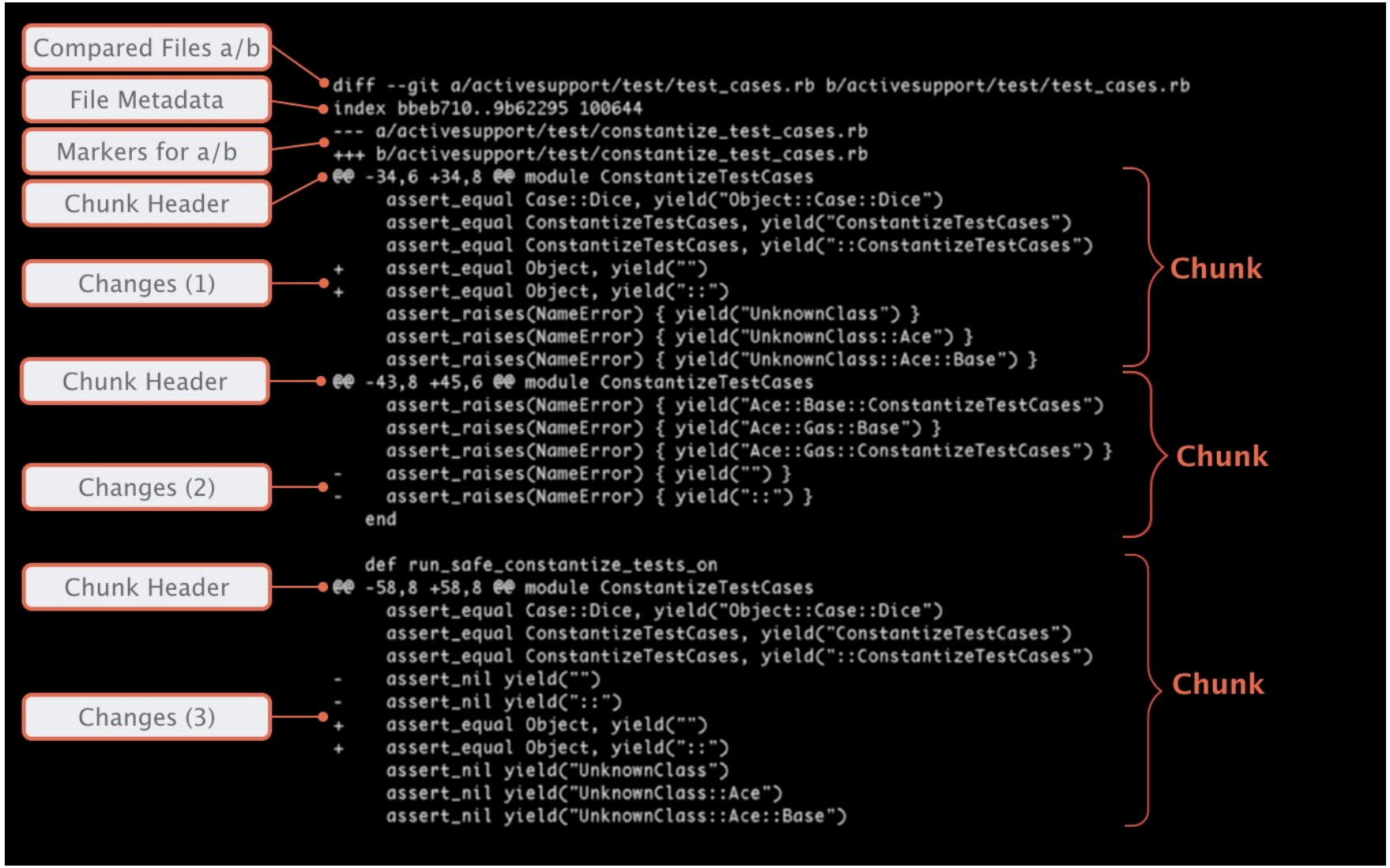
GIT

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



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::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 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

```
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ git diff
diff --git a/index.html b/index.html
index 02ee7de..bea125e 100644
--- a/index.html
+++ b/index.html
@@ -3,7 +3,7 @@
 </head>
 <body>
         <p>
-                This is some sample text
+                I modified this text now
         </p>
 </body>
 </html>
\ No newline at end of file
```

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

CONFLICT RESOLUTION

Merge tools (text-based or GUI based) Often the same tool as Diff

Line-by-line comparison; allows user to choose which lines to keep

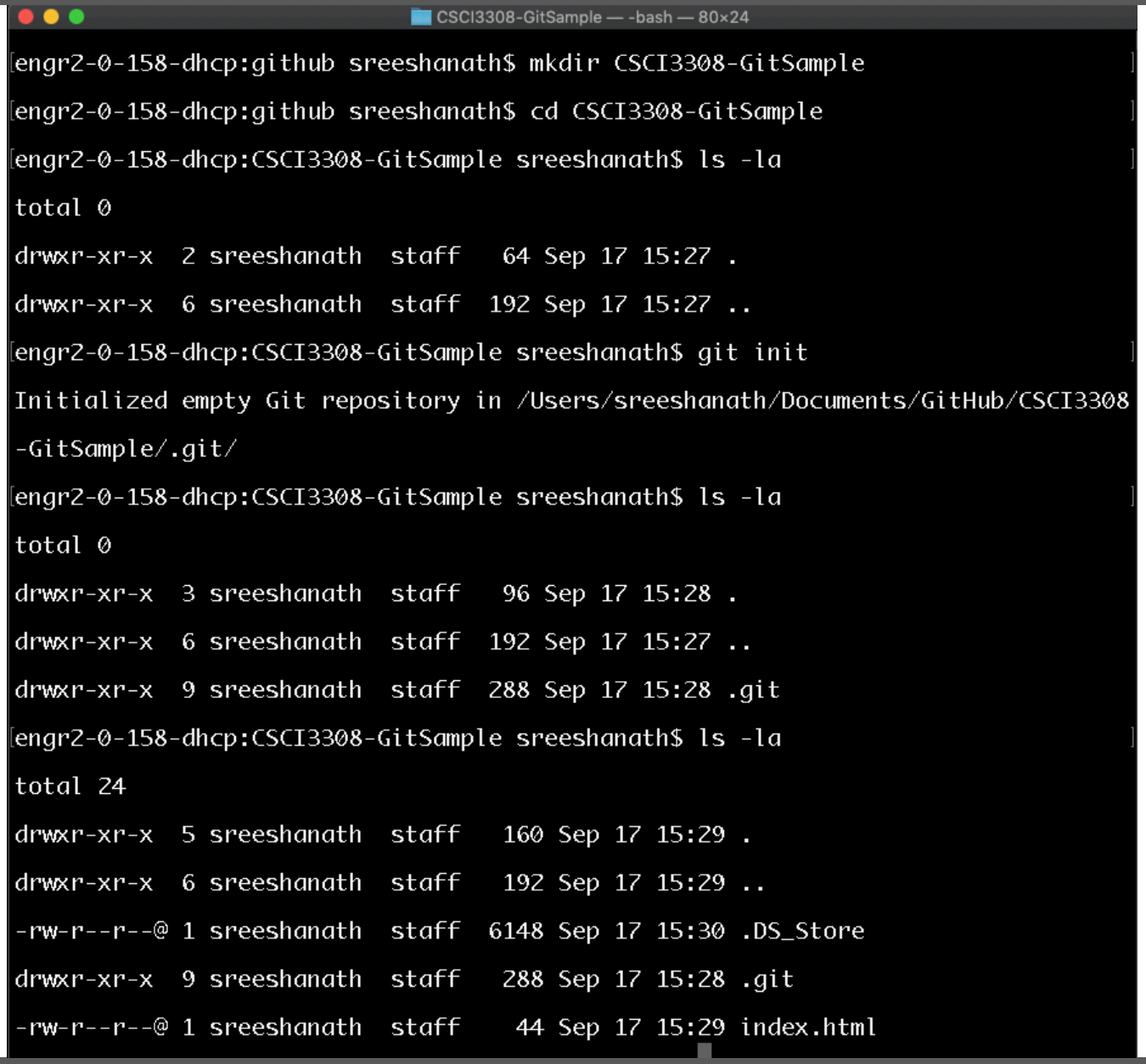
Once complete, “mark resolve” and commit

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


CSCI3308-GitSample — -bash — 80×24
engr2-0-158-dhcp:github sreeshanath$ mkdir CSCI3308-GitSample
engr2-0-158-dhcp:github sreeshanath$ cd CSCI3308-GitSample
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ ls -la
total 0
drwxr-xr-x 2 sreeshanath staff   64 Sep 17 15:27 .
drwxr-xr-x 6 sreeshanath staff 192 Sep 17 15:27 ..
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ git init
Initialized empty Git repository in /Users/sreeshanath/Documents/GitHub/CSCI3308
-GitSample/.git/
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ ls -la
total 0
drwxr-xr-x 3 sreeshanath staff   96 Sep 17 15:28 .
drwxr-xr-x 6 sreeshanath staff 192 Sep 17 15:27 ..
drwxr-xr-x 9 sreeshanath staff 288 Sep 17 15:28 .git
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ ls -la
total 24
drwxr-xr-x 5 sreeshanath staff   160 Sep 17 15:29 .
drwxr-xr-x 6 sreeshanath staff   192 Sep 17 15:29 ..
-rw-r--r--@ 1 sreeshanath staff 6148 Sep 17 15:30 .DS_Store
drwxr-xr-x 9 sreeshanath staff   288 Sep 17 15:28 .git
-rw-r--r--@ 1 sreeshanath staff   44 Sep 17 15:29 index.html

Working Copy

Your Project's Files

tracked

...modified

tracked

...unmodified

untracked

Staging Area

Changes for Next Commit

Local Repo

".git" Folder

```
[engr2-0-158-dhcp:csci3308-gitsample sreeshanath$ git status
On branch newbranch
Your branch is up to date with 'origin/newbranch'.

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

        .DS_Store
        sampletextfile.txt
```

Working Copy

Your Project's Files

tracked

...modified

tracked

...unmodified

untracked

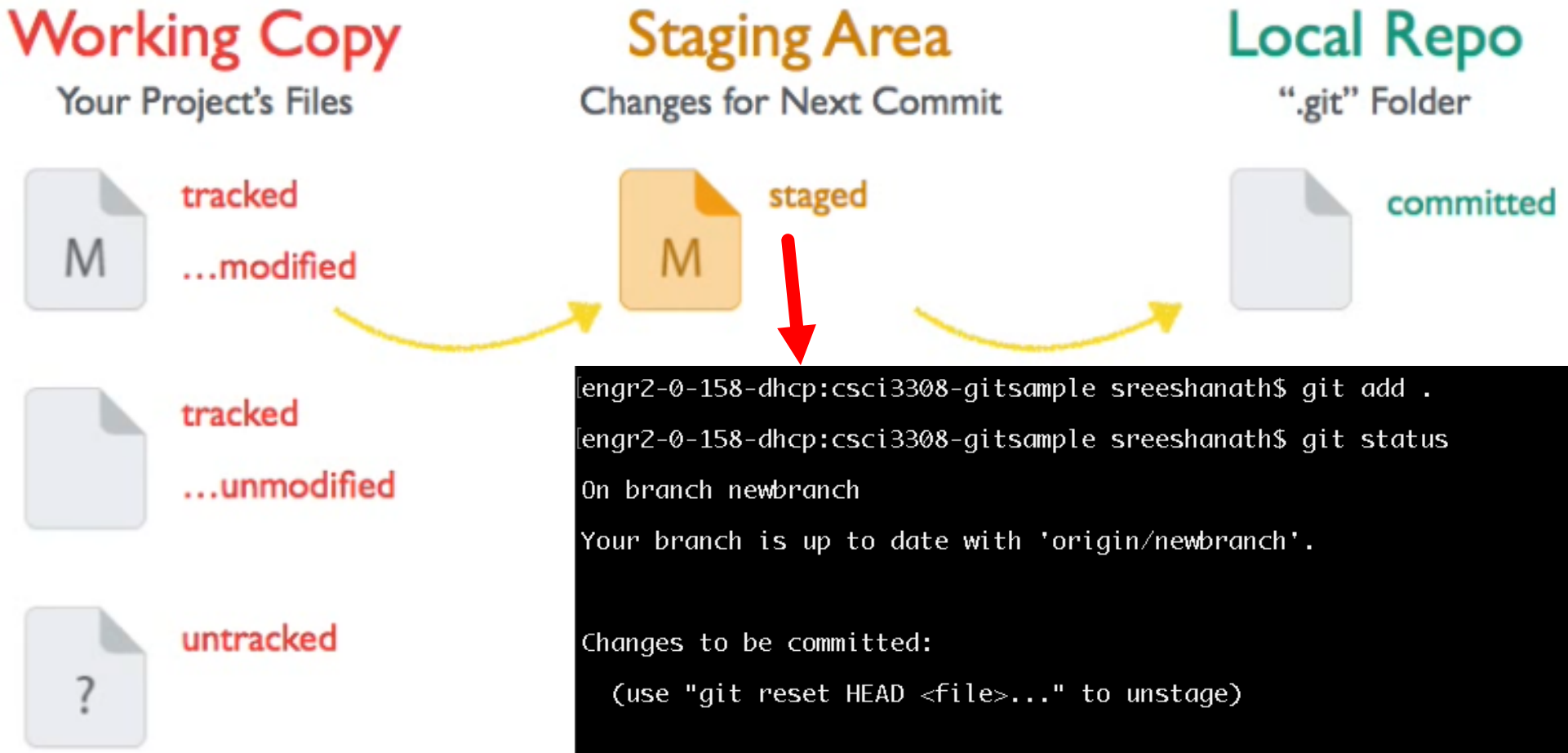
Staging Area

Changes for Next Commit

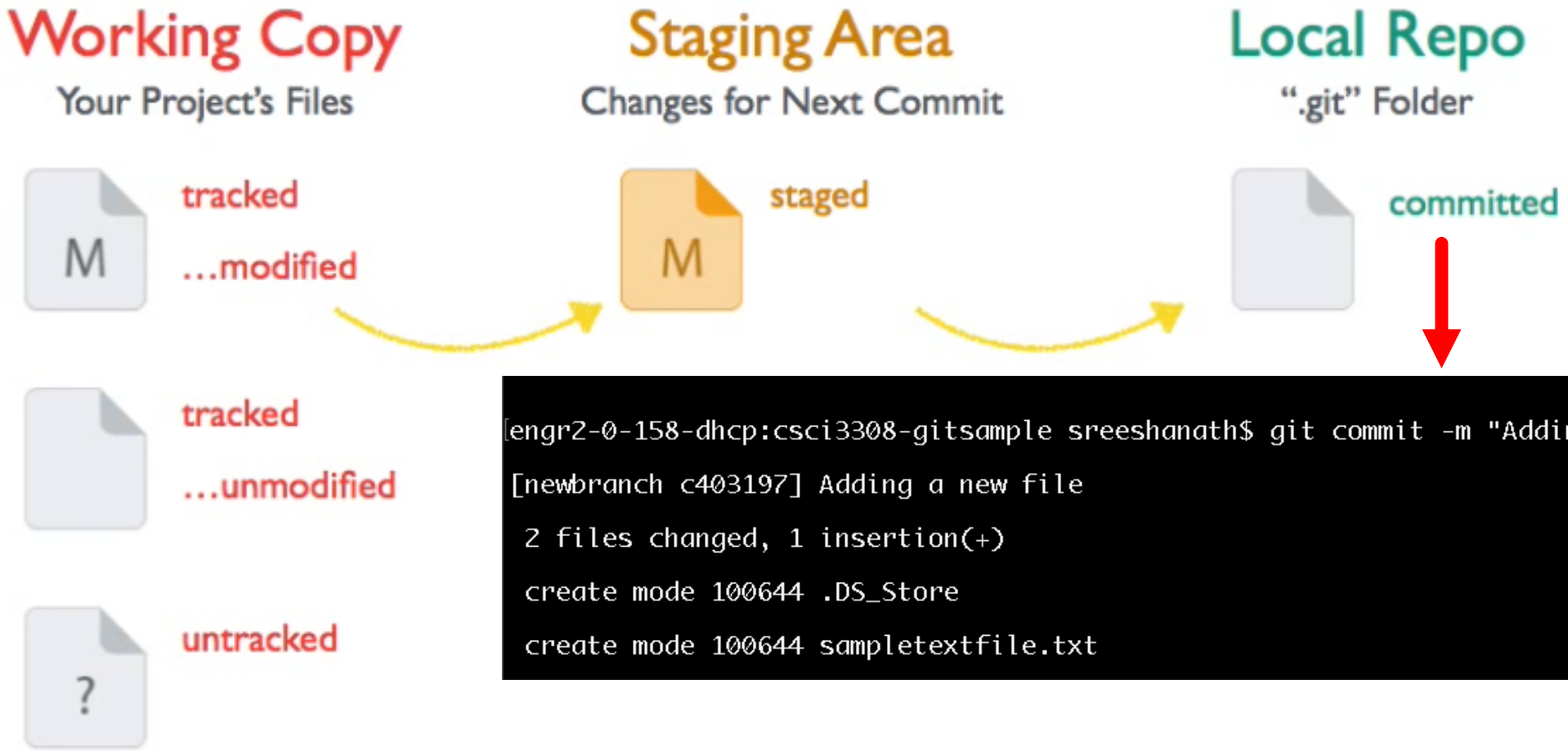
Local Repo

".git" Folder

```
[engr2-0-158-dhcp:csci3308-gitsample sreeshanath$ git status
On branch newbranch
Your branch is ahead of 'origin/newbranch' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   sampletextfile.txt
```




```
engr2-0-158-dhcp:csci3308-gitsample sreeshanath$ git add .
engr2-0-158-dhcp:csci3308-gitsample sreeshanath$ git status
On branch newbranch
Your branch is up to date with 'origin/newbranch'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .DS_Store
        new file:   sampletextfile.txt
```




```
[engr2-0-158-dhcp:csci3308-gitsample sreeshanath$ git commit -m "Adding a new file"
[newbranch c403197] Adding a new file
 2 files changed, 1 insertion(+)
 create mode 100644 .DS_Store
 create mode 100644 sampletextfile.txt
```

```
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ git remote add origin https://g
ithub.com/SreeshaNath/CSCI3308-GitSample.git
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 242 bytes | 242.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/SreeshaNath/CSCI3308-GitSample.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

```
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ git branch newBranch1
engr2-0-158-dhcp:CSCI3308-GitSample sreeshanath$ git branch
* master
  newBranch1
```

LINKS TO ONLINE VIDEO TRAINING

- https://try.github.io (start here)
- https://www.git-tower.com/learn/git/videos (11 free videos)

GIT RECAP

- git help
- git init – prepares your local git
- git clone – copies central repo to local copy (once)
- git add – stages for commit
- 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 updated code from remote repo to local
- git pull – fetch + merge to local (integrates into local)