Builders are Integral to Continuous Integration

<APACHE ANT>
(2000)

GRUNT
(2012)

make
(1976)

Maven™
(2004)

Gradle
(2007)

Gulp
(2016)

Continuous Integration (CI)

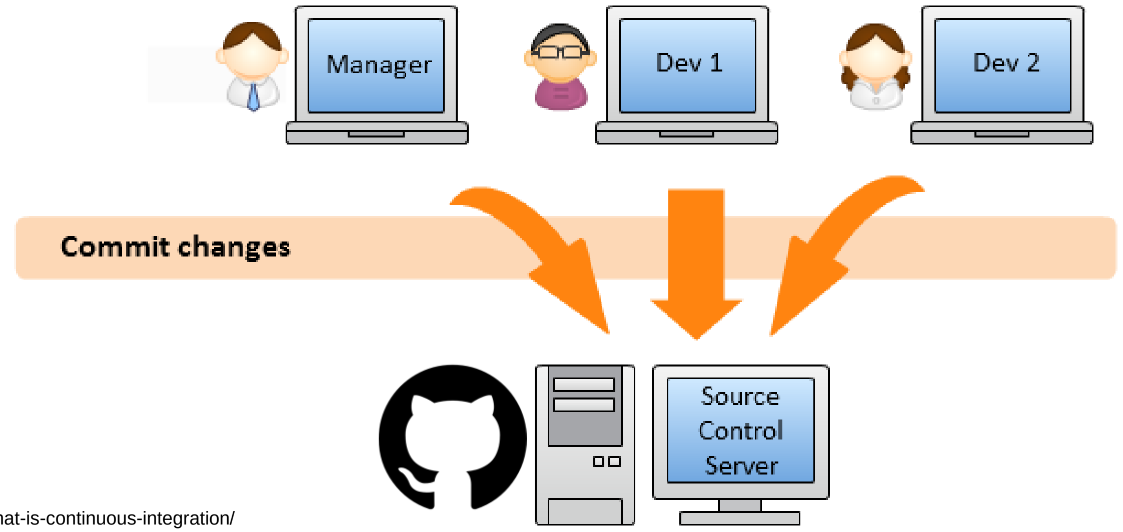
Continuous Integration (CI)

- The practice of merging (i.e., integrating) small code changes to a shared branch as often as possible (at least daily)
  - The goal is to develop and test in smaller increments, rather than at the end of the development cycle (Waterfall)

- Introduced around 1991

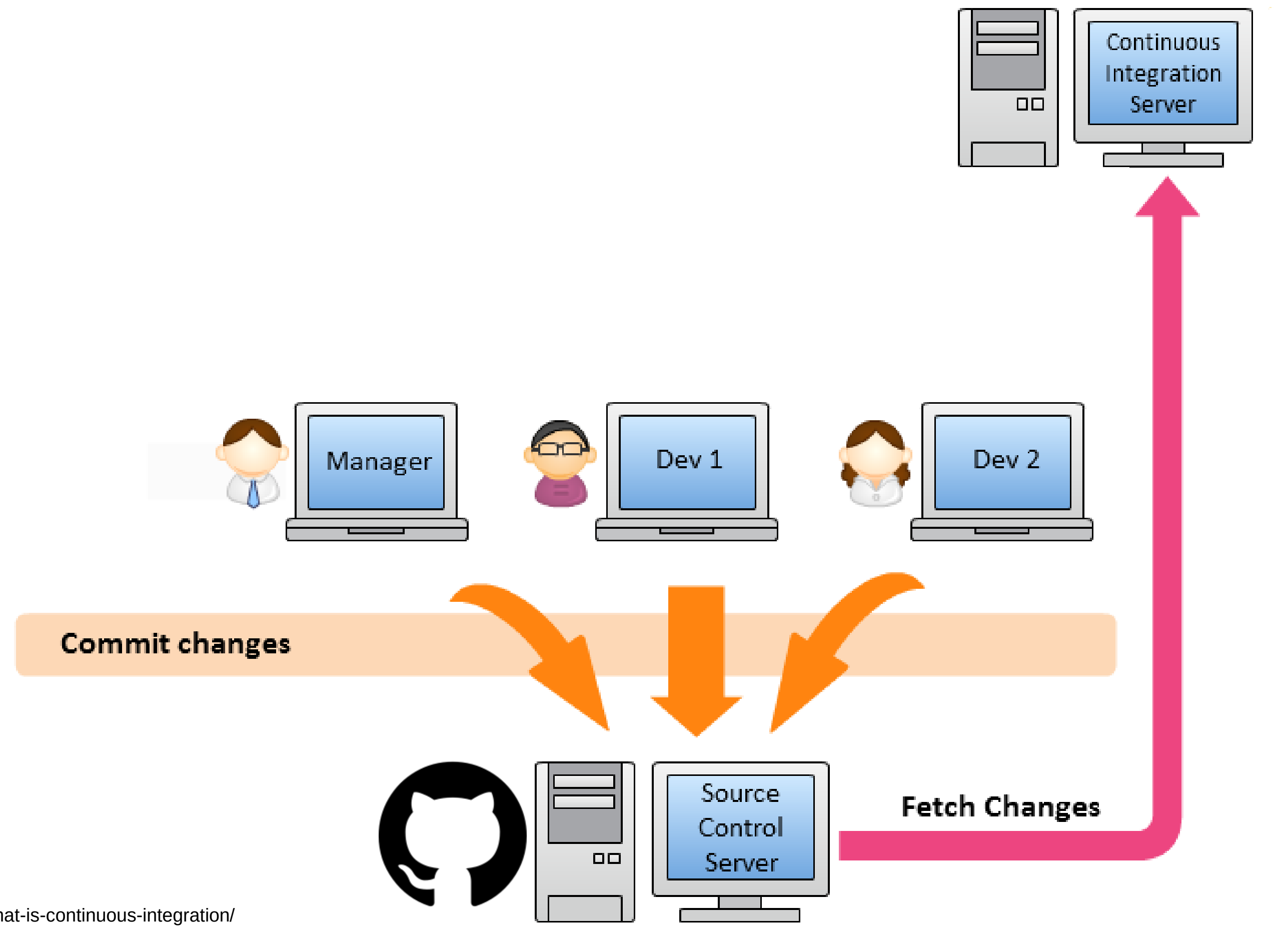
- Adopted by XP (Extreme Programming)

Continuous Integration (CI)



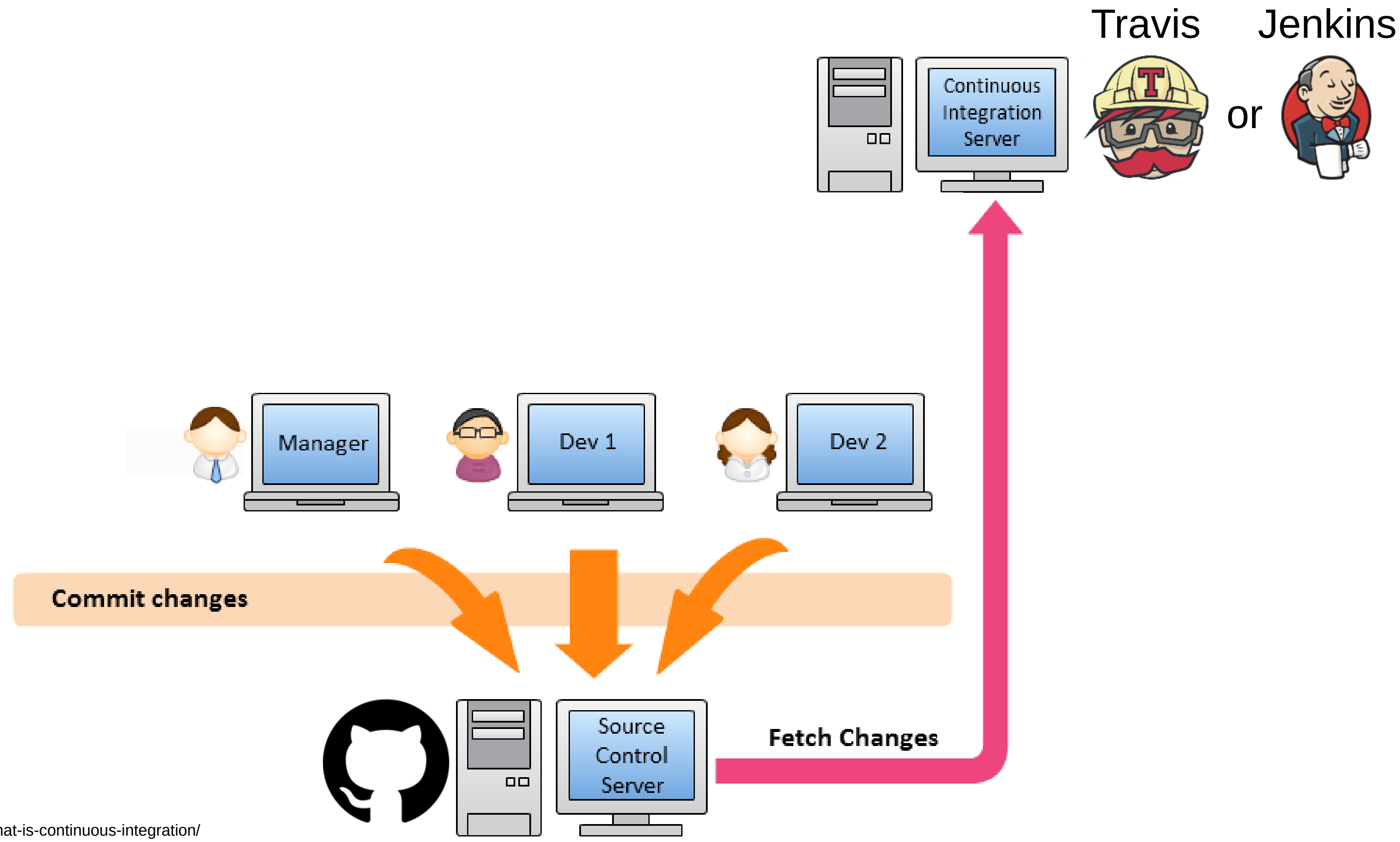
https://www.code-maze.com/what-is-continuous-integration/

Continuous Integration (CI)

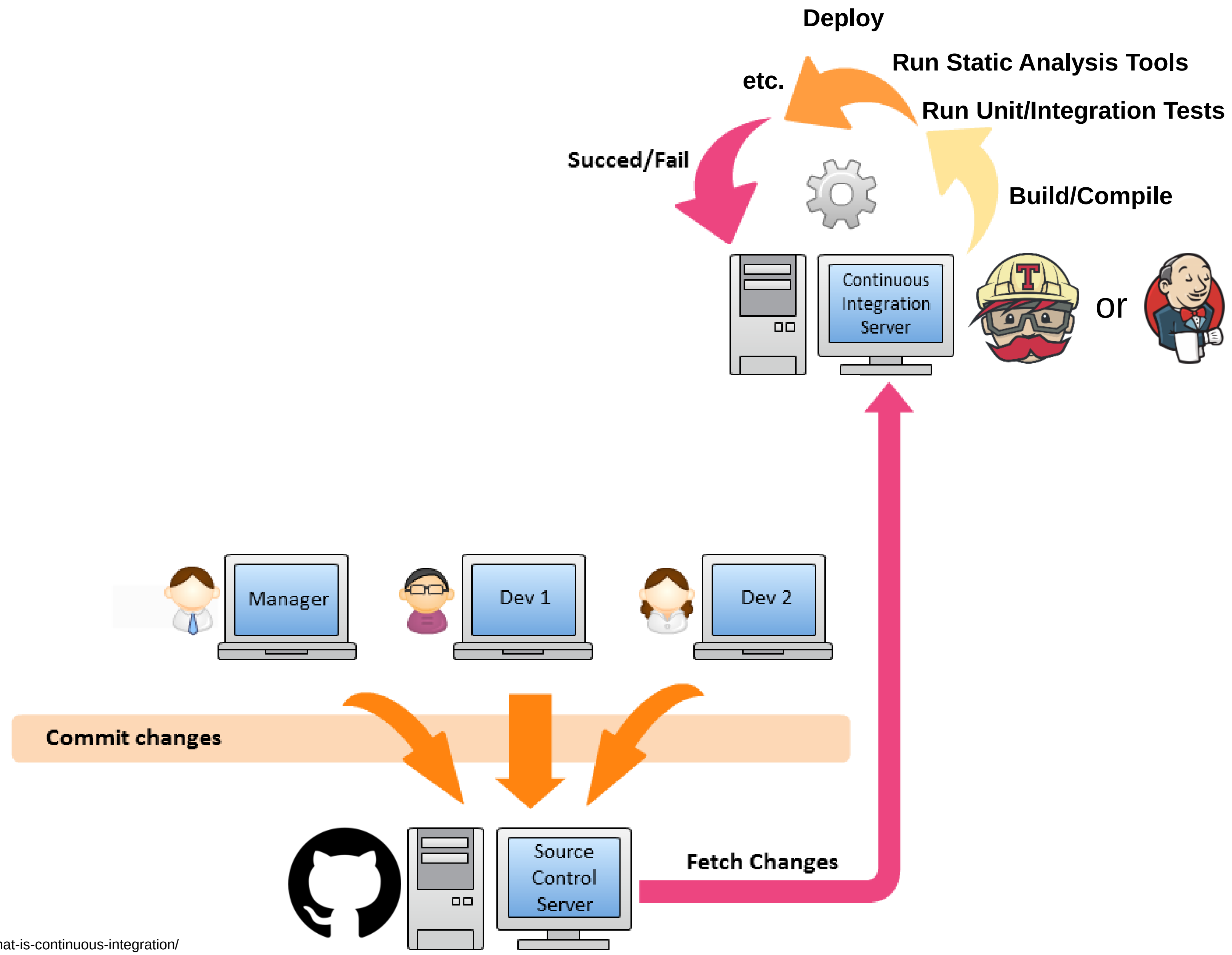


https://www.code-maze.com/what-is-continuous-integration/

Continuous Integration (CI)

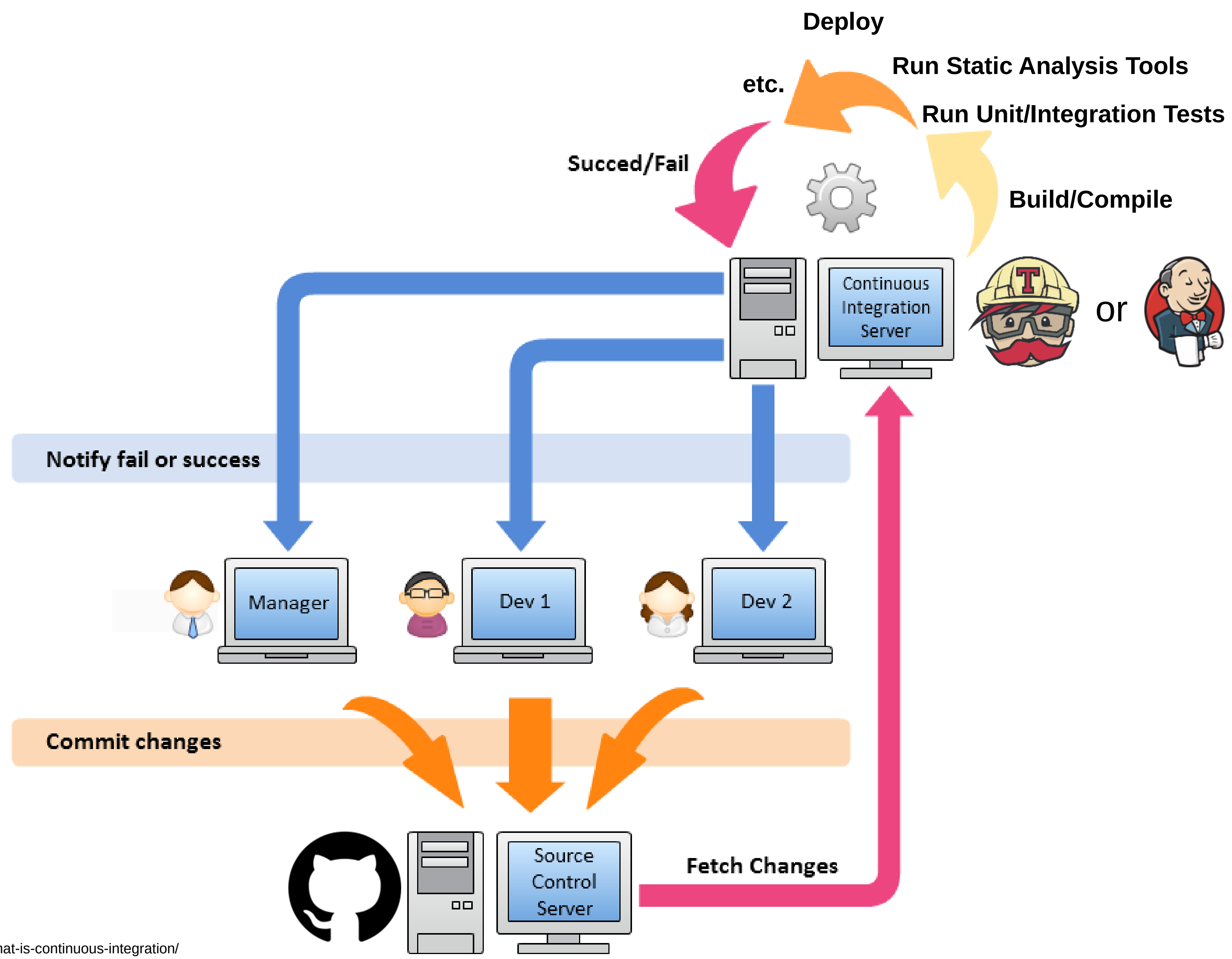


Continuous Integration (CI)



https://www.code-maze.com/what-is-continuous-integration/

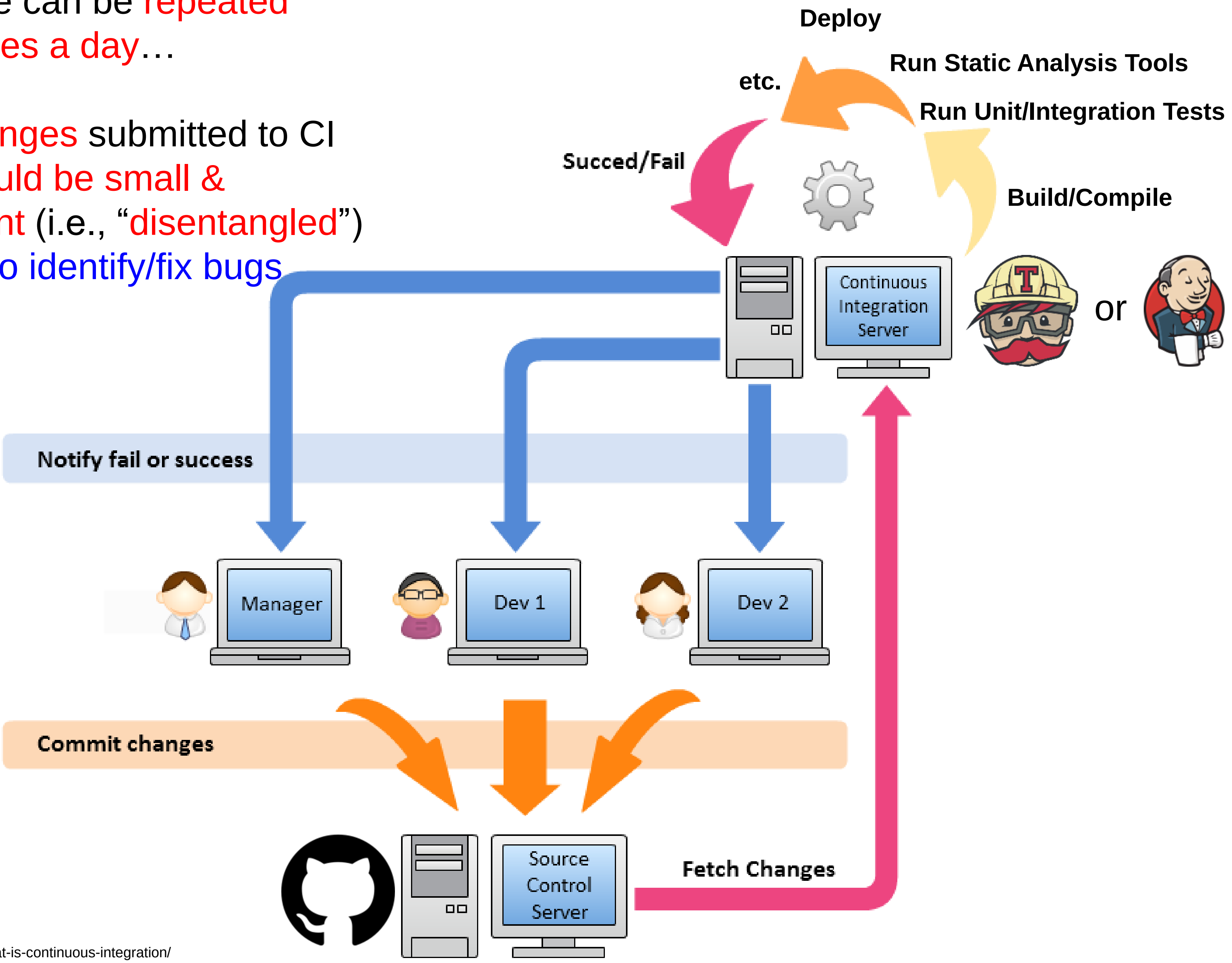
Continuous Integration (CI)



https://www.code-maze.com/what-is-continuous-integration/

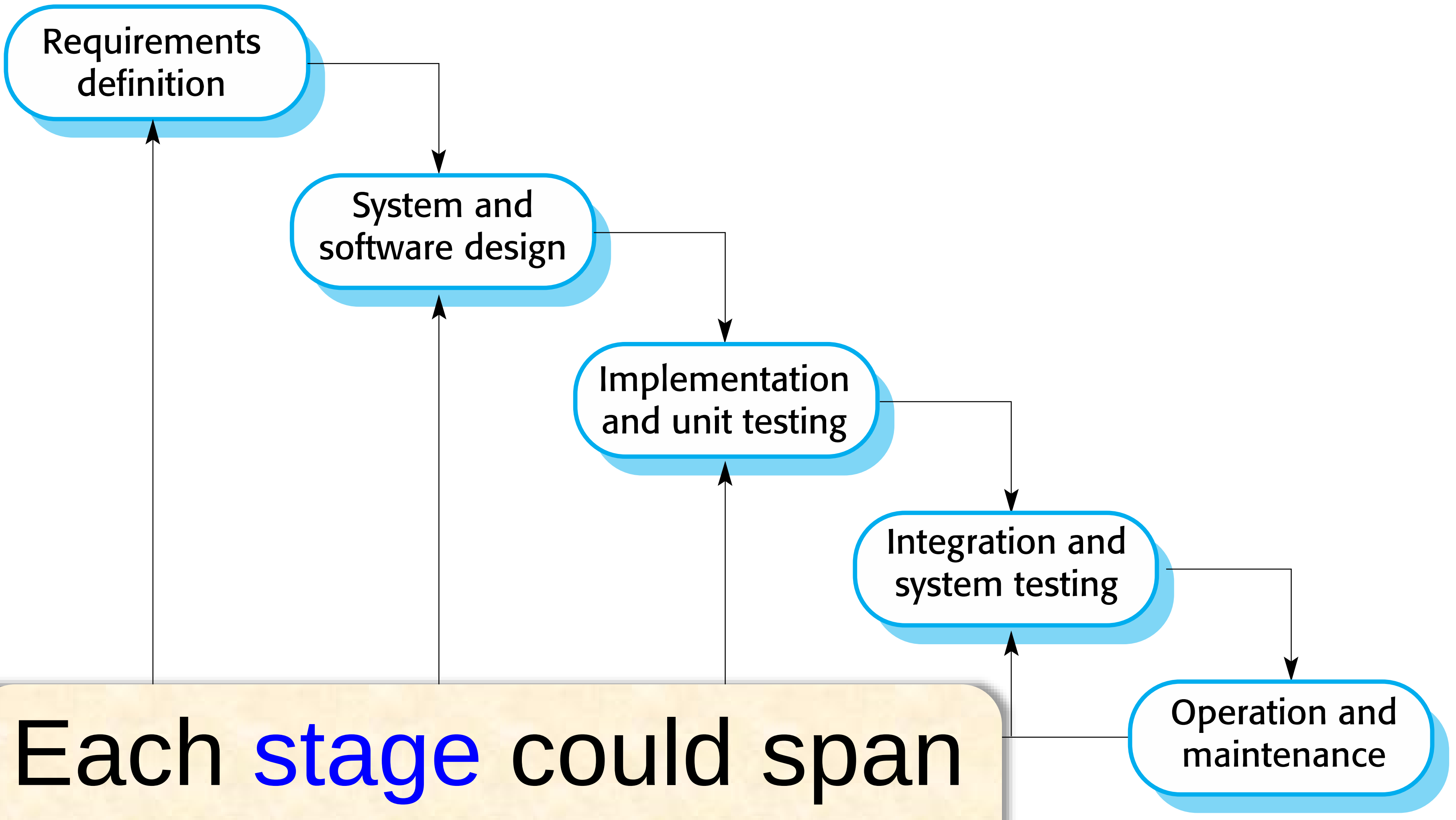
Continuous Integration (CI)

- Entire cycle can be repeated several times a day…
- Ideally changes submitted to CI server should be small & independent (i.e., “disentangled”)
  - Easy to identify/fix bugs



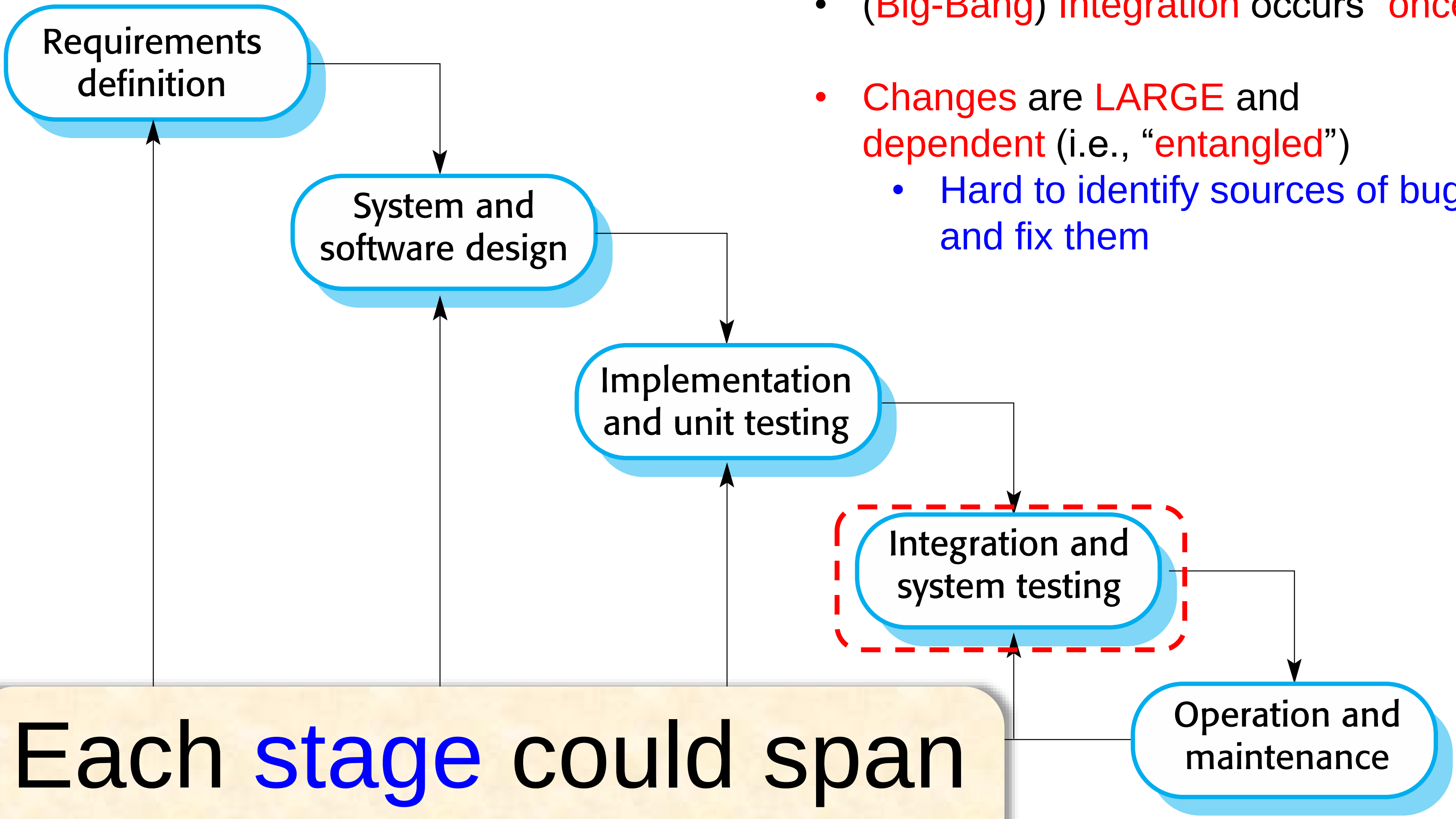
https://www.code-maze.com/what-is-continuous-integration/

CI vs. Waterfall


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Each stage could span for months or years!

CI vs. Waterfall


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

- (Big-Bang) Integration occurs “once”
- Changes are LARGE and dependent (i.e., “entangled”)
  - Hard to identify sources of bugs and fix them

Each stage could span for months or years!

Continuous Integration Advantages

Continuous Integration Advantages

- Integration bugs are detected early
  - minimizes integration risks and
  - avoids "big-bang" integration

- Immediate feedback of system-wide impact of local changes

Continuous Integration Advantages

- Higher code quality
  - less manual effort is spent identifying integration problems
  - automated regression tests “catch” bugs inadvertently introduced in other parts of the system
- Constant availability of a "current" public build (for testing, demo, or release purposes)
  - Increases team morale and confidence in their changes

Continuous Integration Overhead

- Initial infrastructure set-up

- Works best if project has a large suite of tests

Continuous Build (Integration) Services

Continuous Build (Integration) Services

- Product is constructed by a Build Server responsible for:
  - Initiating the build which executes the *builder*
    - At a scheduled time (e.g., nightly builds)
    - Following the completion of another build
    - When a certain action is triggered (e.g., a new commit was submitted to master)
    - Manually started (from a browser)
  - Reporting results of a build to the Team

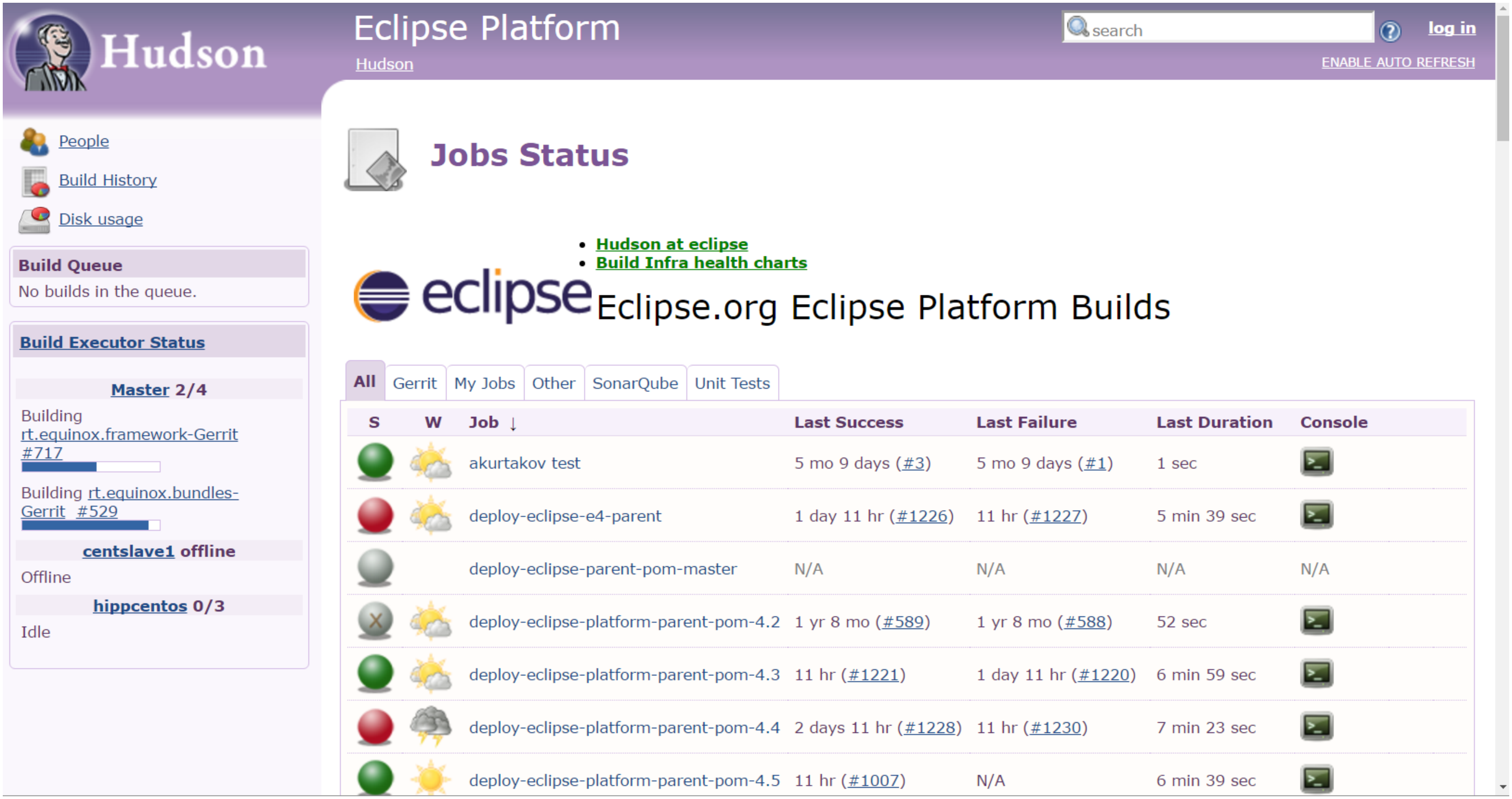
- A *builder* constructs the project binaries

Well-Known Continuous Build Integration Services

Service	**Platform**	**IDEs**	**Builders**
CruiseControl (2001 – 2010)	Cross Platform	Eclipse	Command Line
Hudson (2005)/ Jenkins (2011)	Servlet	Eclipse, NetBeans, etc.	Ant, Maven, Command Line
Team Foundation Server	Windows	Visual Studio, Eclipse	MSBuild
Travis CI	Cloud (GitHub)	?	Language Dependent

Example Build Service:

https://hudson.eclipse.org/



Build Queue

No builds in the queue.

Build Executor Status

Master 2/4

Building rt.equinox.framework-Gerrit #717

Building rt.equinox.bundles-Gerrit #529

centslave1 offline

Offline

hippcentos 0/3

Idle

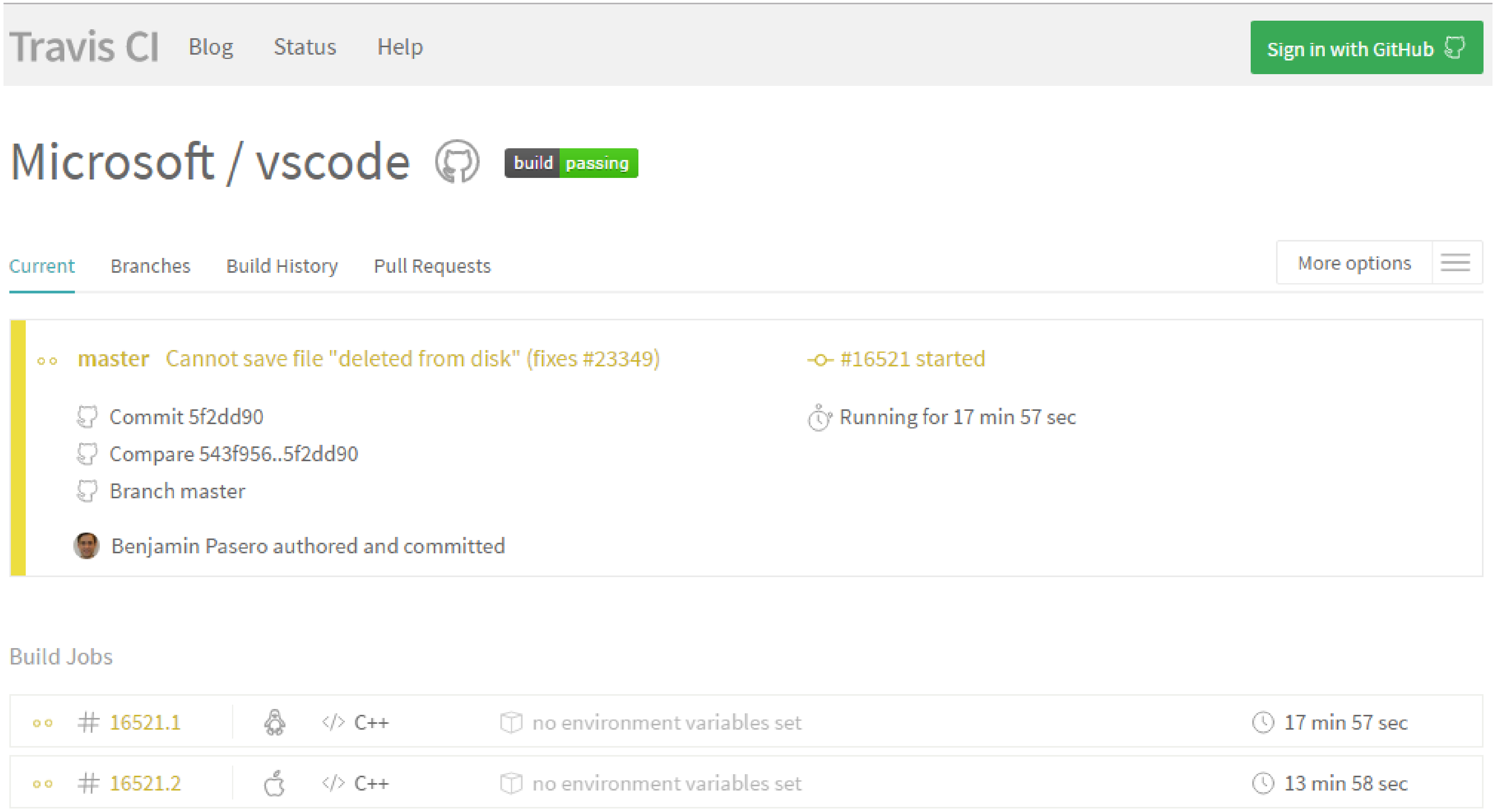
Jobs Status

- Hudson at eclipse
- Build Infra health charts

eclipse Eclipse.org Eclipse Platform Builds

S	W	Job ↓	Last Success	Last Failure	Last Duration	Console
🟢	⛅	akurtakov test	5 mo 9 days (#3)	5 mo 9 days (#1)	1 sec	▣
🔴	⛅	deploy-eclipse-e4-parent	1 day 11 hr (#1226)	11 hr (#1227)	5 min 39 sec	▣
⚪		deploy-eclipse-parent-pom-master	N/A	N/A	N/A	N/A
⚪	⛅	deploy-eclipse-platform-parent-pom-4.2	1 yr 8 mo (#589)	1 yr 8 mo (#588)	52 sec	▣
🟢	⛅	deploy-eclipse-platform-parent-pom-4.3	11 hr (#1221)	1 day 11 hr (#1220)	6 min 59 sec	▣
🔴	⛈	deploy-eclipse-platform-parent-pom-4.4	2 days 11 hr (#1228)	11 hr (#1230)	7 min 23 sec	▣
🟢	☀	deploy-eclipse-platform-parent-pom-4.5	11 hr (#1007)	N/A	6 min 39 sec	▣

Example Build Service:
https://travis-ci.org/Microsoft/vscode



Travis CI   Blog   Status   Help   Sign in with GitHub

Microsoft / vscode   build passing

Current   Branches   Build History   Pull Requests   More options

master   Cannot save file "deleted from disk" (fixes #23349)   #16521 started

Commit 5f2dd90   Running for 17 min 57 sec

Compare 543f956..5f2dd90

Branch master

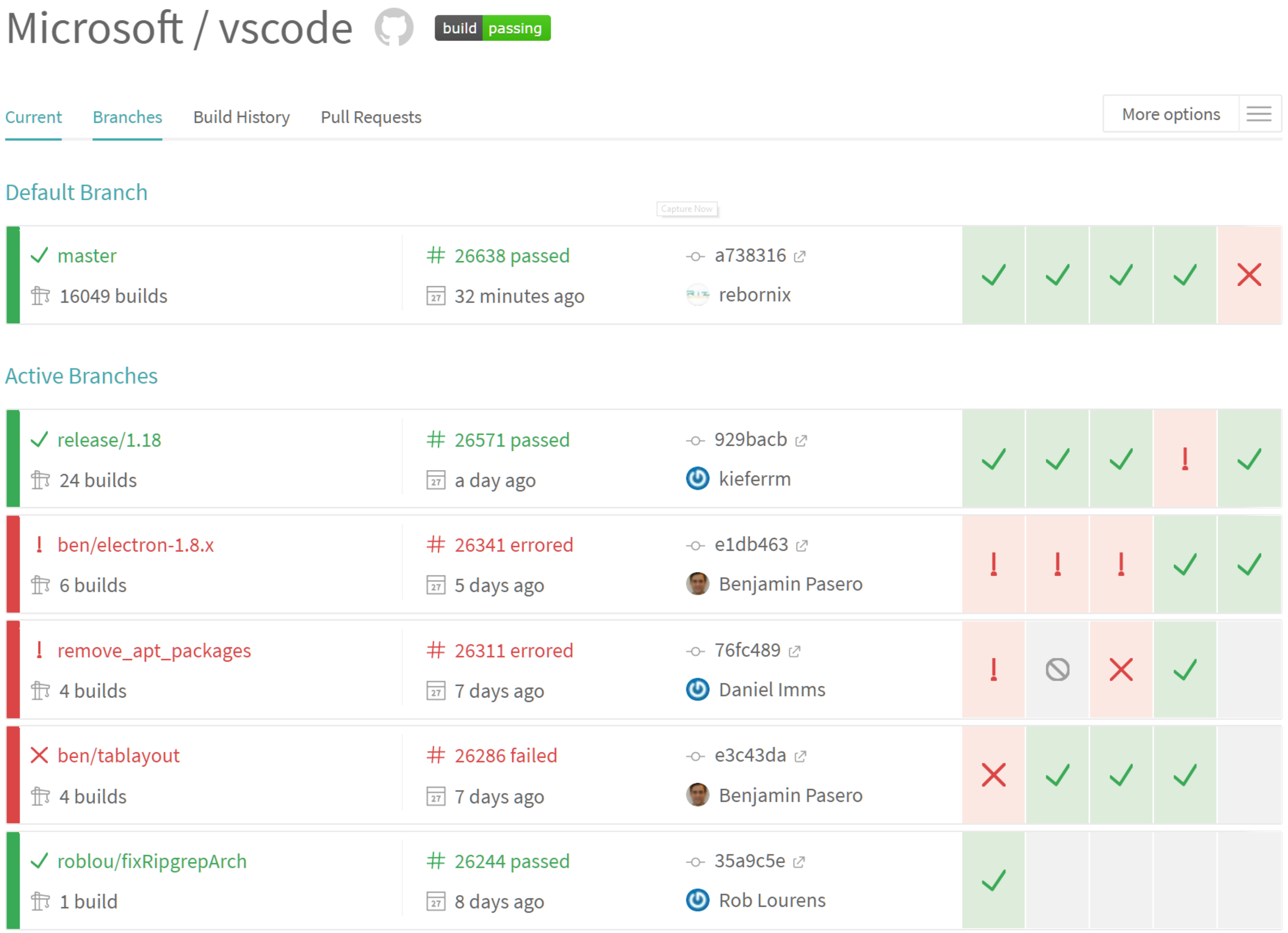
Benjamin Pasero authored and committed

Build Jobs

# 16521.1		C++	no environment variables set	17 min 57 sec
# 16521.2		C++	no environment variables set	13 min 58 sec

Example Build Service on Multiple Branches:

https://travis-ci.org/Microsoft/vscode/branches



Microsoft / vscode build passing

Current Branches Build History Pull Requests More options

Default Branch

Capture Now

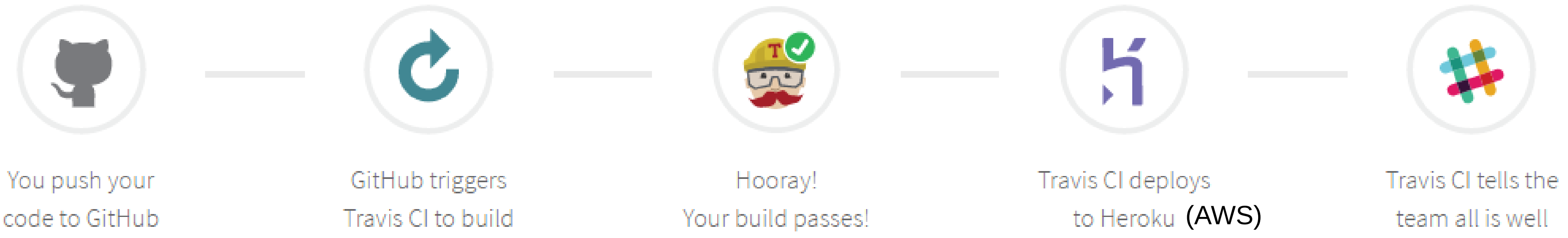
Branch	Builds	Last build	When	Commit	Author					
✓ master	16049 builds	# 26638 passed	32 minutes ago	a738316	rebornix	✓	✓	✓	✓	✗

Active Branches

Branch	Builds	Last build	When	Commit	Author					
✓ release/1.18	24 builds	# 26571 passed	a day ago	929bacb	kieferrm	✓	✓	✓	!	✓
! ben/electron-1.8.x	6 builds	# 26341 errored	5 days ago	e1db463	Benjamin Pasero	!	!	!	✓	✓
! remove_apt_packages	4 builds	# 26311 errored	7 days ago	76fc489	Daniel Imms	!	⃠	✗	✓	
✗ ben/tablayout	4 builds	# 26286 failed	7 days ago	e3c43da	Benjamin Pasero	✗	✓	✓	✓	
✓ roblou/fixRipgrepArch	1 build	# 26244 passed	8 days ago	35a9c5e	Rob Lourens	✓				

Travis Workflows Summary

Branch build flow



You push your code to GitHub

GitHub triggers Travis CI to build

Hooray! Your build passes!

Travis CI deploys to Heroku (AWS)

Travis CI tells the team all is well

Travis Workflows Summary

Branch build flow

You push your code to GitHub — GitHub triggers Travis CI to build — Hooray! Your build passes! — Travis CI deploys to Heroku (AWS) — Travis CI tells the team all is well

Pull-Request Build Flow

A pull request is created — GitHub tells Travis CI the build is mergeable — Hooray! Your build passes! — All is well Travis CI updates the PR that it passed — You merge in the PR goodness

Team Workflows

Workflow Needs

- Each developer needs some one-on-one time with the code to be able to work independently

Workflow Needs

- Each developer needs some one-on-one time with the code to be able to work independently
  - I/You need to add features and/or fix defects in the code
  - I'm/You're probably going to break the code for a while
  - I/You don't want to see your / my changes while you're / I'm debugging them

Workflow Needs

- Each developer needs some one-on-one time with the code to be able to work independently

Workflow Needs

- Each developer needs some one-on-one time with the code to be able to work independently

- We also need an easy and safe way to integrate our changes (i.e., when I’ve finished my changes, I need to test them with those made by other Developers)

Workflow Needs

- Each developer needs some one-on-one time with the code to be able to work independently

- We also need an easy and safe way to integrate our changes (i.e., when I’ve finished my changes, I need to test them with those made by other Developers)
  - Ensure my changes are compatible with their changes
  - I need to do this in a safe place so, if it goes poorly, the other Developers will still have a stable code base
  - When I’m sure that my changes work with those made by others, then I need to publish my code changes for their use

Workflow Needs

- Each developer needs some one-on-one time with the code to be able to work independently
- We also need an easy and safe way to integrate our changes (i.e., when I've finished my changes, I need to test them with those made by other Developers)
- Don't want to lose the source code ⇒ Version Control

Version Control

Version Control

- We must never, never, never lose the source code
  - Sometimes, despite our best efforts, we introduce changes that break the code
  - And we need to get back to an earlier, stable version
  - Occasionally my hard drive crashes

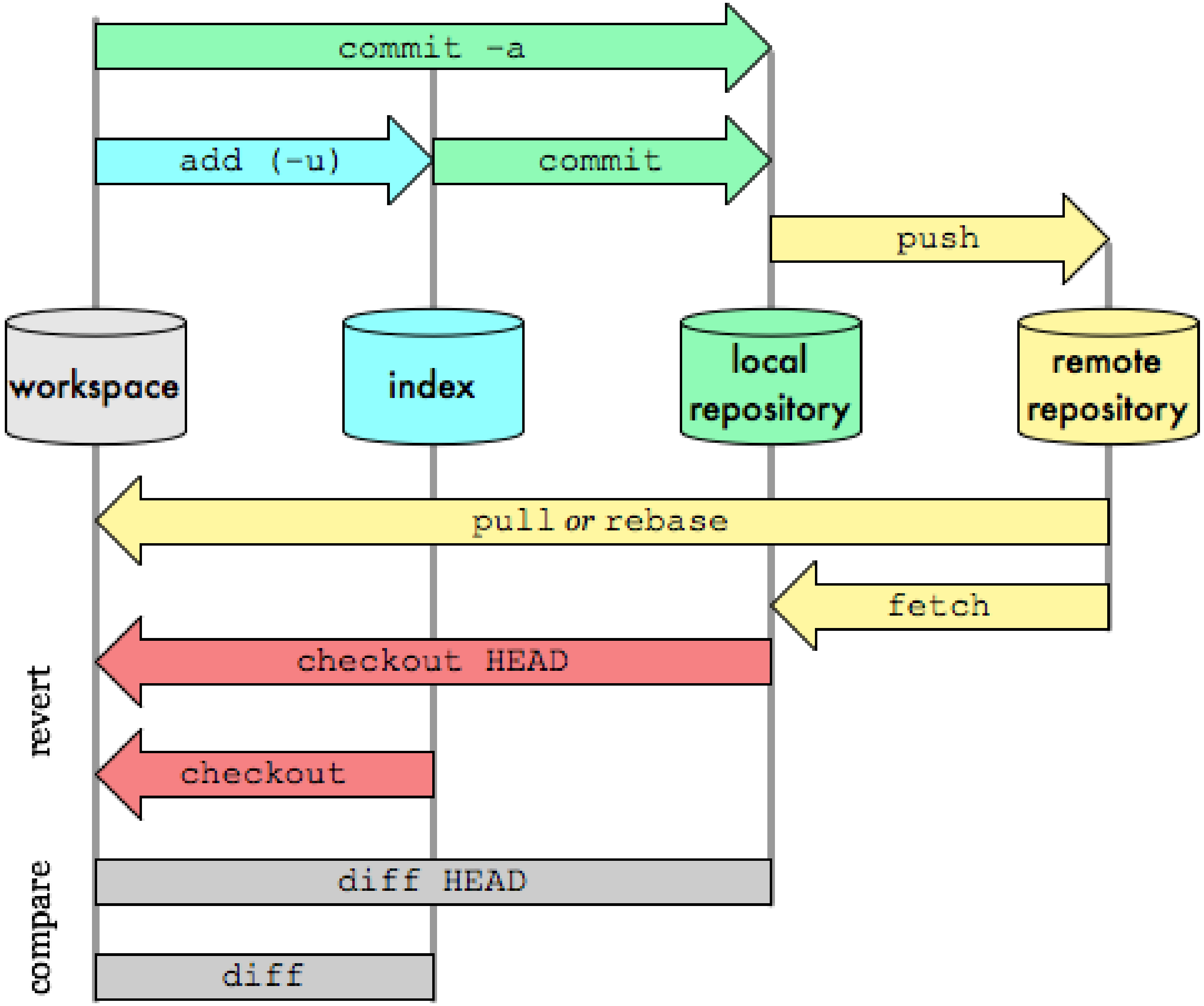
- We need the ability to recall any version of the source code

Version Control

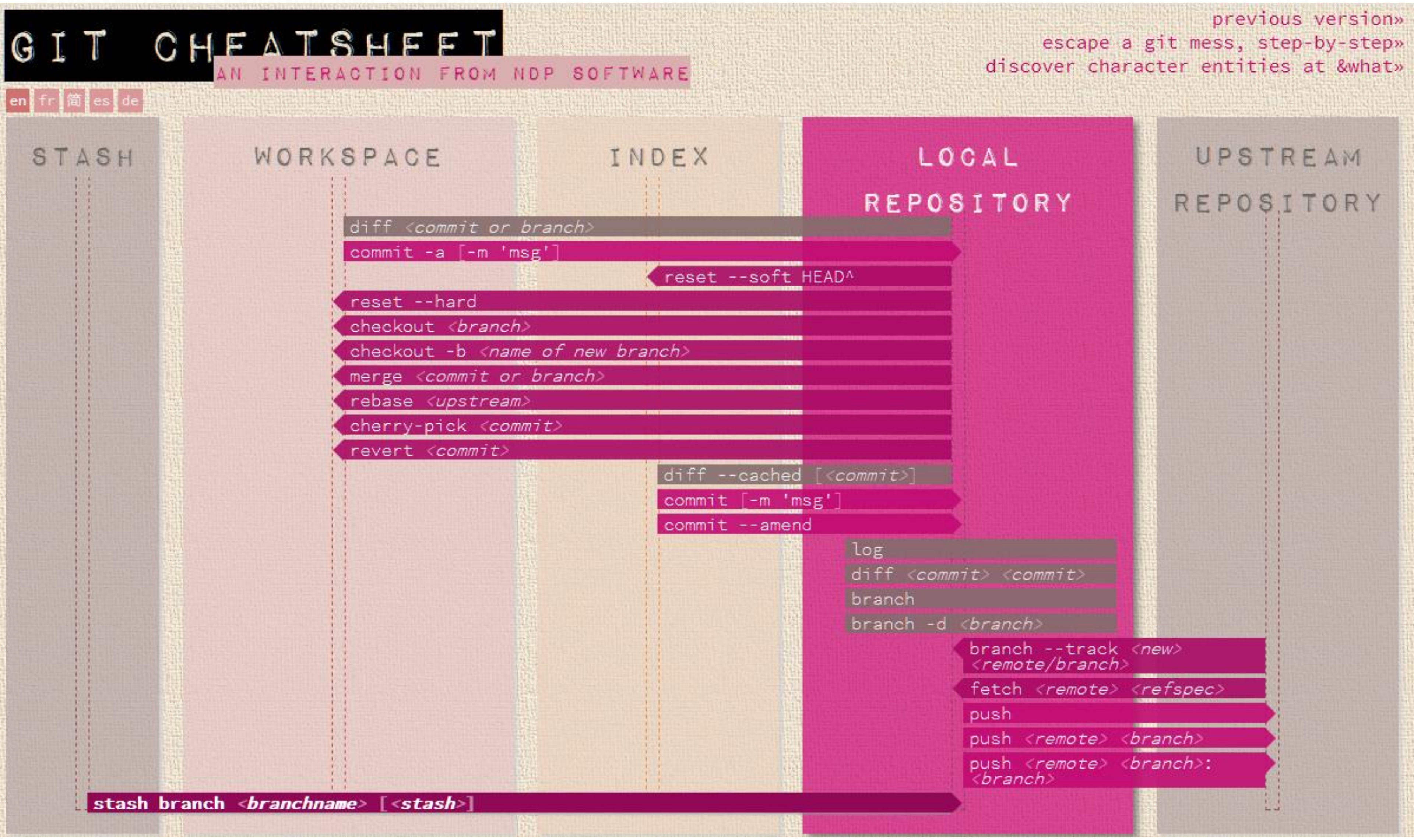
- We want tools to illustrate the changes between versions
  - Which files changed?
  - What did they change in those files?
  - Who made that change!?

Git Data Transport Commands

http://osteele.com



Git Resources


GIT CHEATSHEET
AN INTERACTION FROM NDP SOFTWARE
previous version»
escape a git mess, step-by-step»
discover character entities at &what»
en fr 简 es de
STASH
WORKSPACE
INDEX
LOCAL REPOSITORY
UPSTREAM REPOSITORY

```
diff <commit or branch>
commit -a [-m 'msg']
reset --soft HEAD^
reset --hard
checkout <branch>
checkout -b <name of new branch>
merge <commit or branch>
rebase <upstream>
cherry-pick <commit>
revert <commit>
diff --cached [<commit>]
commit [-m 'msg']
commit --amend
log
diff <commit> <commit>
branch
branch -d <branch>
branch --track <new> <remote/branch>
fetch <remote> <refspec>
push
push <remote> <branch>
push <remote> <branch>:<branch>
stash branch <branchname> [<stash>]
```

http://ndpsoftware.com/git-cheatsheet.html

Git Resources

- https://www.atlassian.com/git/tutorials/syncing
- CS-HU 250 Introduction to Version Control
- https://www.lynda.com/Git-tutorials/Git-Essential-Training/100222-2.html
  - Lynda courses are free with Boise Public Library card (which is also free)
  - http://www.boisepubliclibrary.org/research-learning/

Git: An Example Source Code Management System

- Repository: A copy of the complete history of the project
  - .git folder

- Remote Repository: Usually an "official" location where the team agrees to host their integrated product (e.g., GitHub)

Git: An Example Source Code Management System

- Working Tree: The folder where you modify, create and (sometimes) delete the project's files (e.g. *.java) on your computer. Also known as the *working directory* or the *workspace* or the *sandbox*.
  - Important: Files in a repository are managed by git — you only edit files in the *workspace*.

Branches

- A branch is a place where the Developers integrate their changes to the code
  - Two or three Developers might create a *story branch* to integrate all the Tasks contributing to a specific user story
  - Or a Team might create a *sprint branch* to integrate all the user stories contributing to a specific sprint. The Team would likely have their Continuous Integration system build/test this branch.

Branches

- Branches control the visibility of changes to the code
- Developers working on a specific branch see those changes
- Developers working on other branches do not see them

Branches in Git

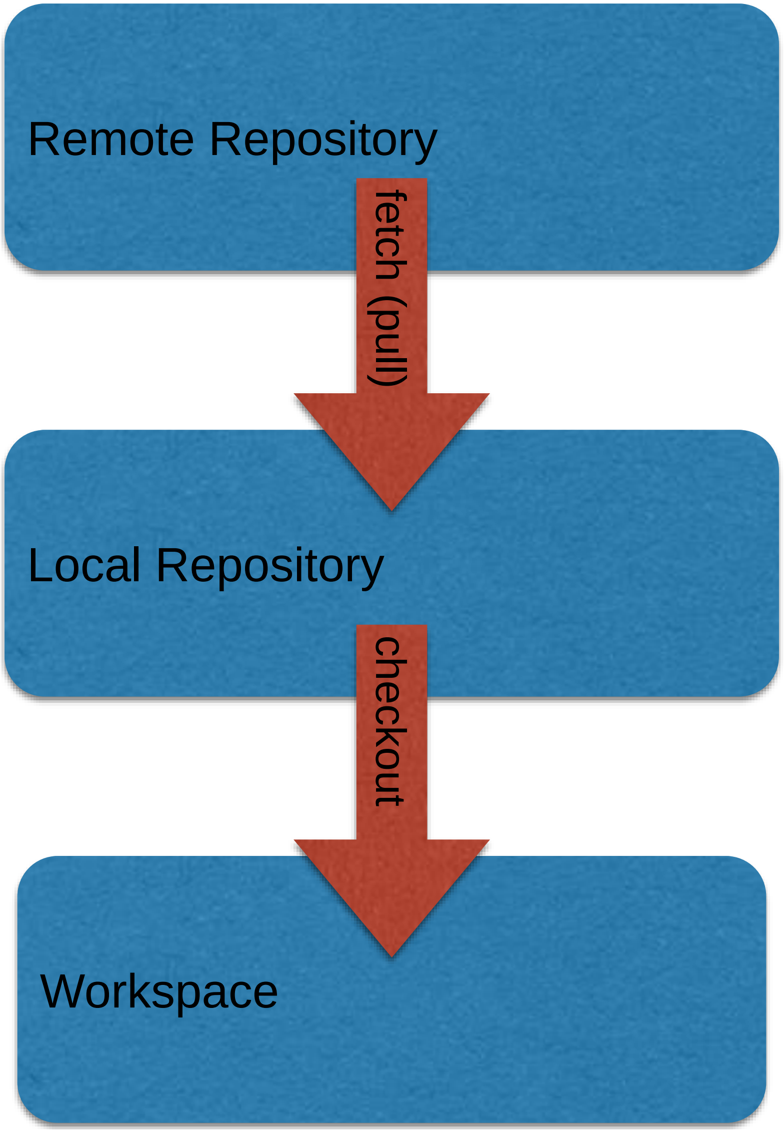
- Every repository has at least one branch, the *master* branch
- Most teams will choose to create multiple branches
- Branches can be created, merged and deleted
- A *merge* integrates the changes from another branch

Obtaining a Copy of a Remote Repository

- Prereq: Install and configure git on your computer

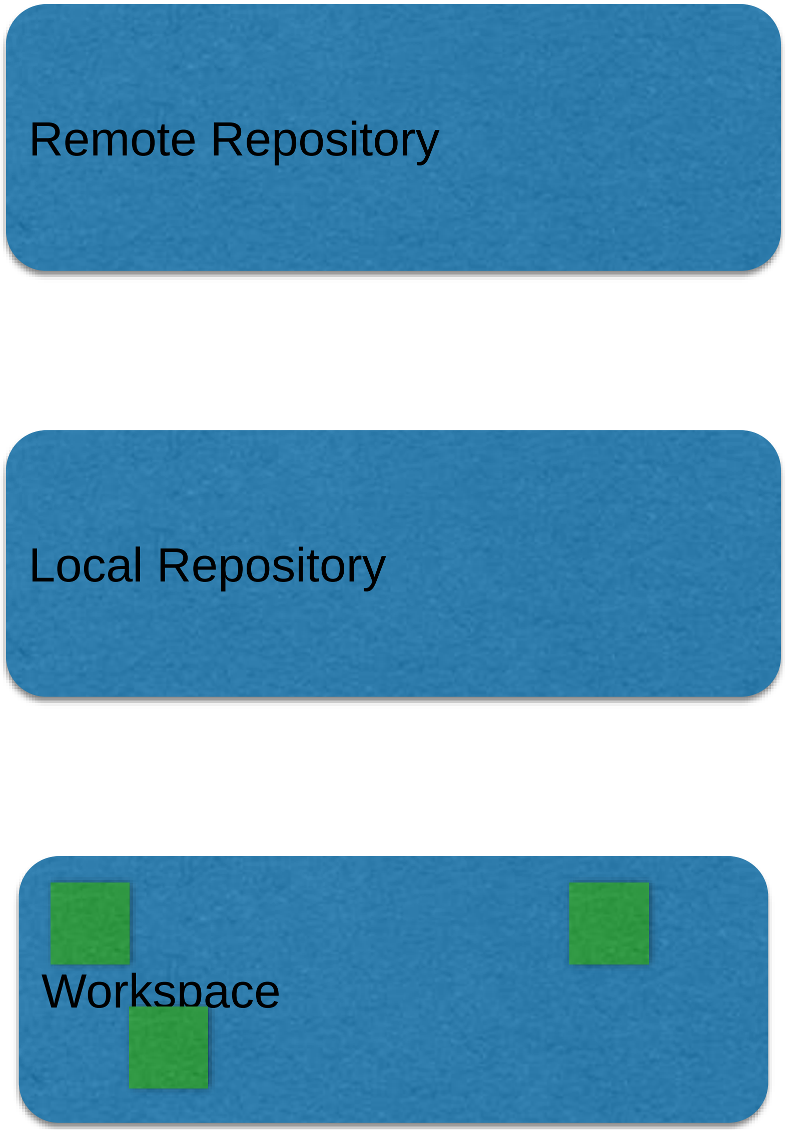
```
git clone <repository>
```

- Downloads an existing repository to your computer
- Checkout the master branch to your workspace
- After which you can edit the project's files



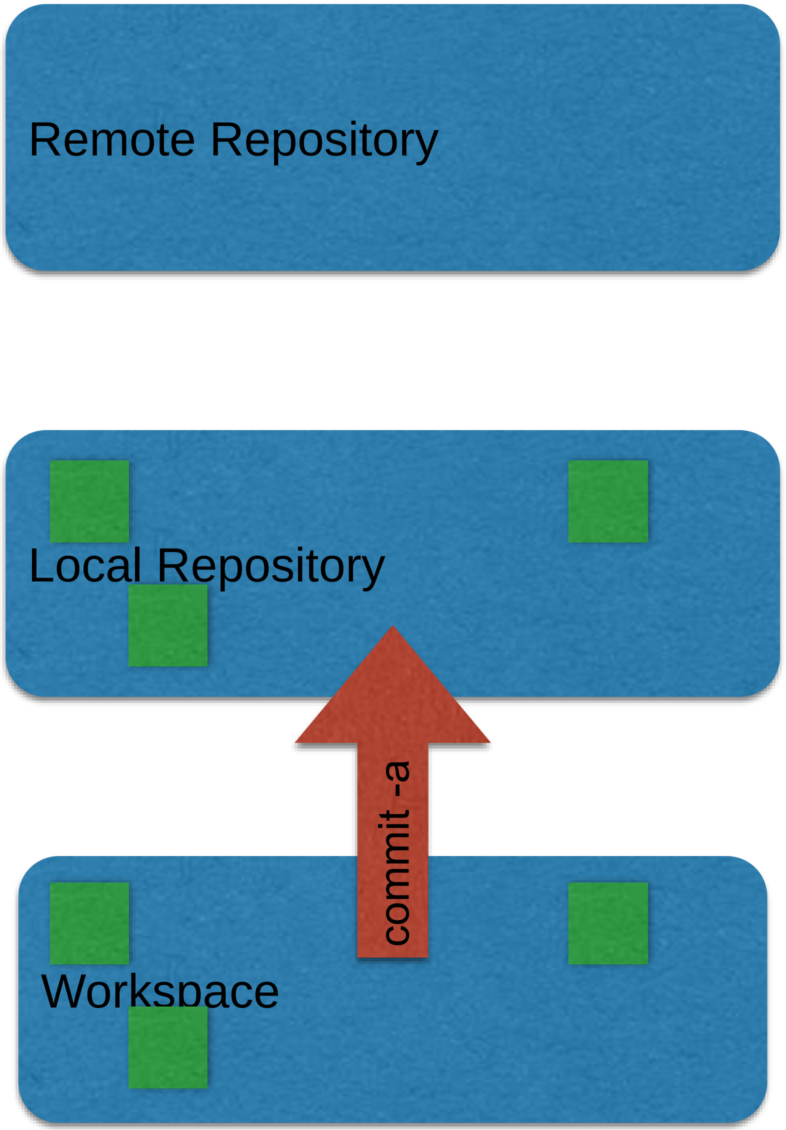
Making Changes

- Your edits occur only in your own, private workspace
- No one else but you can see your changes
- Git doesn't track changes in your workspace
- So they won't appear in a file's history
- And you won't see other Developers' changes



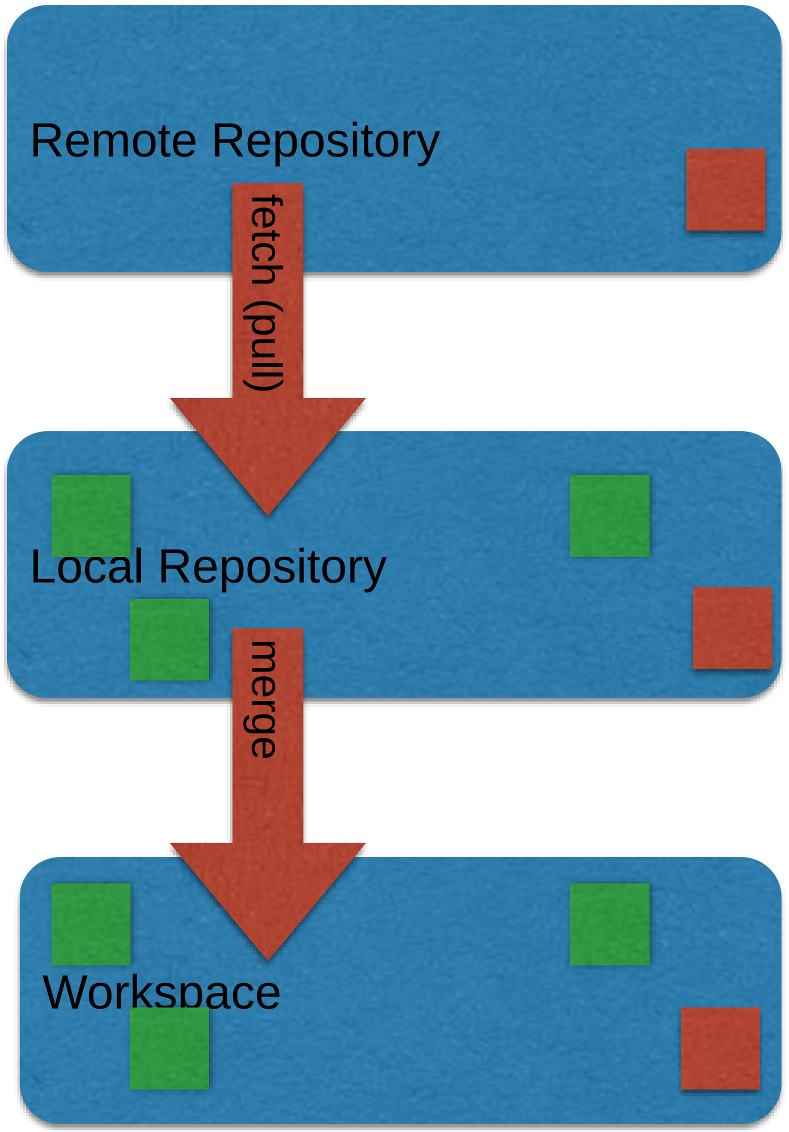
Committing Your Changes

- When you wish git to track your changes in the project history
- git commit -a -m "msg explaining these changes"
- Adds your changes to git's *index* (a staging area)
- And commits those changes to the local repository
- Other Developers still won't see your changes
- But you can recover a committed version of a file



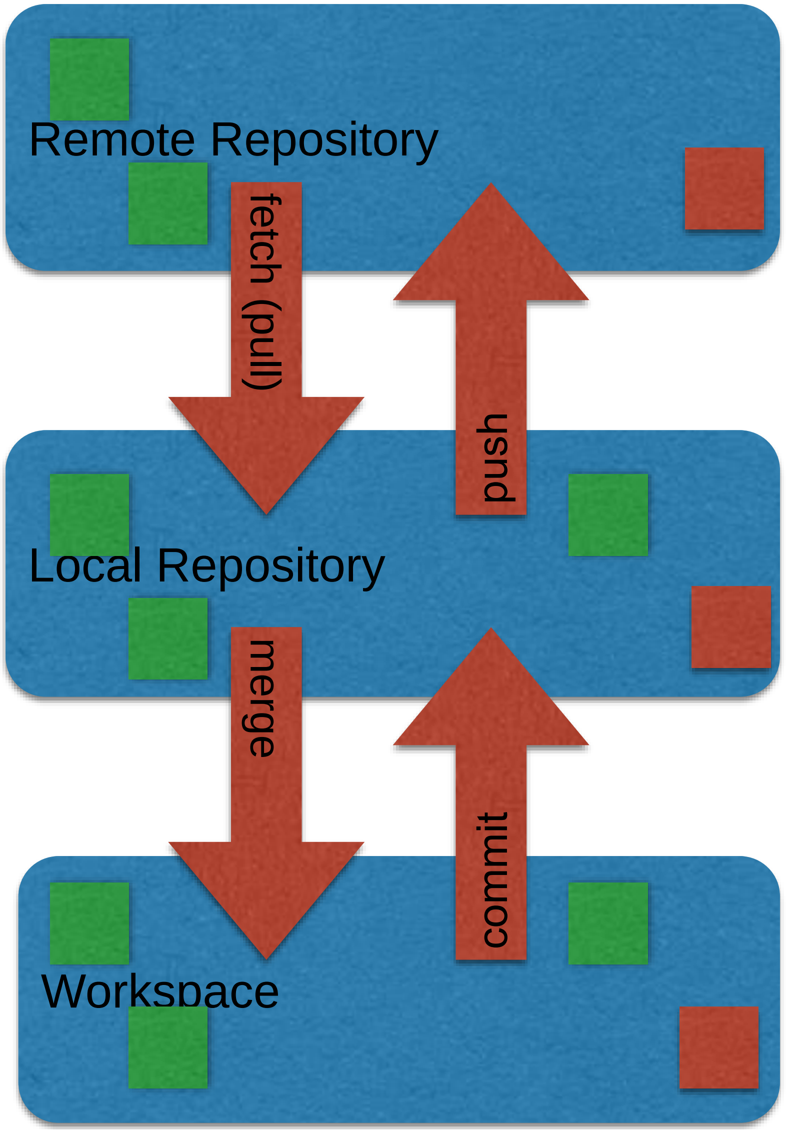
Sharing (Sync) Your Changes

- After you have tested your changes in the workspace and you are ready to share them with other Developers
- Step 1: git pull
  - Fetch changed files from the remote repository to your local repository
  - Merge those changes into your workspace



Sharing (Sync) Your Changes

- After you have tested your changes in the workspace and you are ready to share them with other Developers
- Step 1: git pull
  - Fetch changed files from the remote repository to your local repository
  - Merge those changes into your workspace
- Step 2: git push
  - Updates the remote repository with your changes



What is this *merge*?

- While you were editing files in your workspace…
- Another Developer edited files in their workspace as well
- And they pushed their changes to the remote repository
- So before you can push your changes, you must first merge their changes into your changes
- To do this, git pull (= git fetch + git merge)
  - will first fetch their changes into your local repository
  - And then will merge their changes into your workspace
- And finally, if the merge is successful, commit your workspace

Merge Conflicts

- An automatic merge can fail
- If, for example, both you and the other Developer modified the same lines in the same file, then you will encounter a *merge conflict*
- Git cannot automatically resolve *merge conflicts*
- You must resolve the issue manually
- See: https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/

Merge Conflicts

- Merge conflicts are a major pain if they involve numerous files!
  - Keep number of files/changes to a minimum in a commit
  - Keep commits small (few lines of code changed)

Merge Conflicts

- Avoid unnecessary merge conflicts, e.g.,:
  - if you reformat all your *.java files to correct indentations
  - or brace placement…
  - or change spaces to tabs or vice versa…
  - or change the LF ("\n") character to CRLF ("\r\n")…
  - you might see hundreds… even thousands of merge conflicts

- e.g., Use checkstyle
  http://checkstyle.sourceforge.net/

Merging Branches

- You don’t need a branch to work independently — your computer’s workspace is wholly independent of all others
- You need a branch to collaborate on something
  - Like a feature or a user story
  - Or a sprint
  - Or a hot fix for a released product
- You’ll likely want to integrate your code with that of other developers on a branch
- You’ll likely want your CI server to build and test on one or more branches

Merging Branches

- If your project has a lot of branches…
  - You'll likely encounter merge conflicts
  - Because two or more branches changed the same lines in the same files

- Branches that are "out" for a long time often encounter more conflicts

- Solution?

Merging Branches

- If your project has a lot of branches…
  - You'll likely encounter merge conflicts
  - Because two or more branches changed the same lines in the same files

- Branches that are "out" for a long time often encounter more conflicts
  - Many teams seek ways to keep branches short-lived
  - Detailed User Stories can help shorten branch-life
  - "Constantly" sync the branch with its parent (e.g., rebase)