

Version 1.5  
09/20/23



# Prerequisites

- Have a recent version of Git downloaded and running on your system
  - For Windows, suggest using Git Bash Shell interface
- If you don't already have one, sign up for free GitHub account at  
<http://www.github.com>
- Labs docs are in <https://github.com/skilldocs/git4>
- Labs doc for workshop

<https://github.com/skilldocs/git4/blob/main/git4-4-labs.pdf>



# Git in 4 Weeks

## Part 4

Presented by

Brent Laster

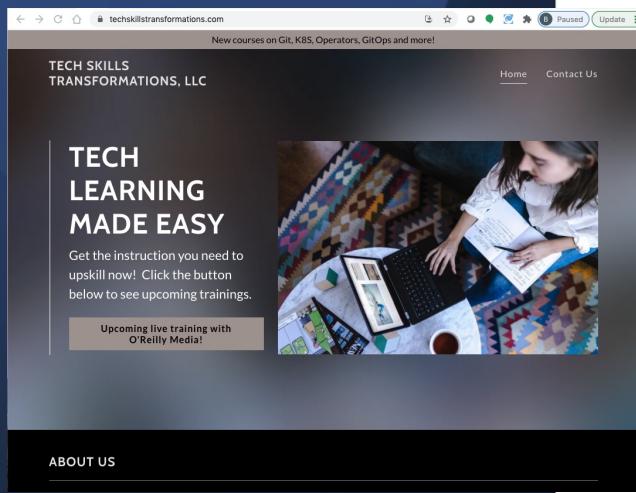
Tech Skills Transformations LLC

© 2023 Brent C. Laster & Tech Skills Transformations LLC



All rights reserved

# About me



- Founder, Tech Skills Transformations LLC
- R&D DevOps Director
- Global trainer – training (Git, Jenkins, Gradle, CI/CD, pipelines, Kubernetes, Helm, ArgoCD, operators)
- Author -
  - Professional Git
  - Jenkins 2 – Up and Running book
  - Learning GitHub Actions
  - Various reports on O'Reilly Learning
- <https://www.linkedin.com/in/brentlaster>
- @BrentCLaster
- GitHub: brentlaster



# Professional Git Book

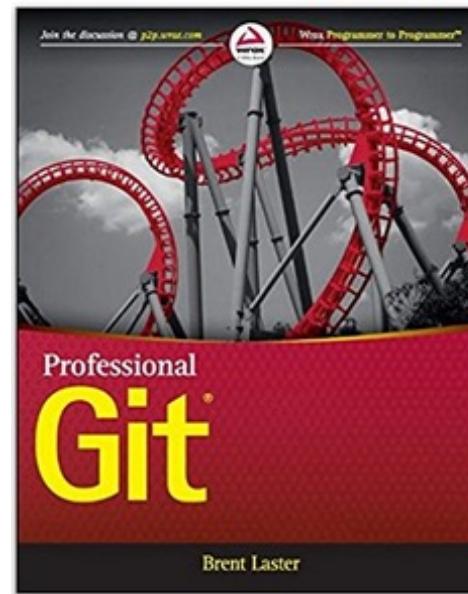
- Extensive Git reference, explanations,
- and examples
- First part for non-technical
- Beginner and advanced reference
- Hands-on labs

## Professional Git 1st Edition

by [Brent Laster](#) (Author)

7 customer reviews

[Look inside](#)



© 2023 Brent C. Laster &

Tech Skills Transformations LLC



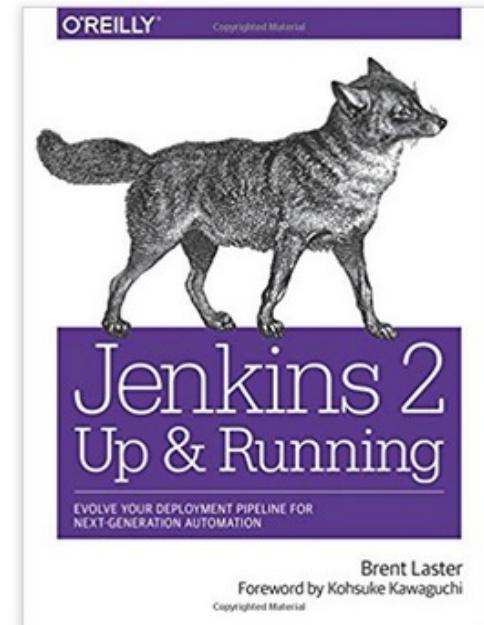
# Jenkins 2 Book

- Jenkins 2 – Up and Running
- “It’s an ideal book for those who are new to CI/CD, as well as those who have been using Jenkins for many years. This book will help you discover and rediscover Jenkins.” *By Kohsuke Kawaguchi, Creator of Jenkins*

★★★★★ 5 customer reviews

#1 New Release in Java Programming

[Look inside](#) ↴





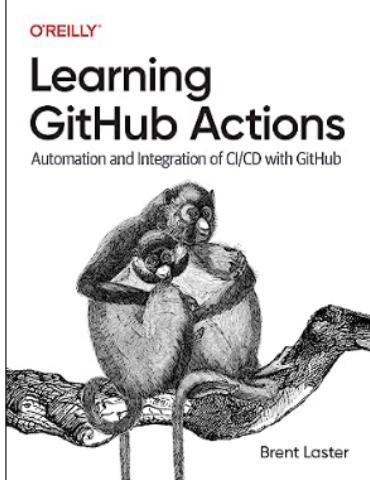
# GitHub Actions book

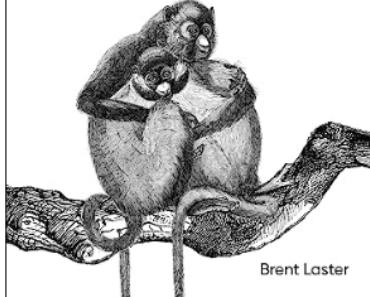
All Get the app Prime Day Back to School Add People Buy Again Gift Cards Recommendations IT Supplies Business Savings Amazon Basics EN Hello, Account Group: Tech Skills Transfor

Guide buyers in your org Books Advanced Search New Releases Best Sellers & More Children's Books Textbooks Textbook Rentals Best Books of the Month Your Company Bookshelf

 Watch now +

Books > Computers & Technology > Programming



O'REILLY  
**Learning GitHub Actions**  
Automation and Integration of CI/CD with GitHub  
  
Brent Laster

Roll over image to zoom in

Follow the Author

 Brent Laster Follow

**Paperback**  
**\$65.99** ✓prime

1 New from \$65.99

Pre-order Price Guarantee. [Terms](#)

Automate your software development processes with GitHub Actions, the continuous integration and continuous delivery platform that integrates seamlessly with GitHub. With this practical book, open source author, trainer, and DevOps director Brent Laster explains everything you need to know about using and getting value from GitHub Actions. You'll learn what actions and workflows are and how they can be used, created, and incorporated into your processes to simplify, standardize, and automate your work in GitHub.

This book explains the platform, components, use cases, implementation, and integration points of actions, so you can leverage them to provide the functionality and features needed in today's complex pipelines and software development processes. You'll learn how to design and implement automated workflows that respond to common events like pushes, pull requests, and review updates. You'll understand how to use the components of the GitHub Actions platform to gain maximum automation and benefit.

With this book, you will:

- Learn what GitHub Actions are, the various use cases for them, and how to incorporate them into your processes
- Understand GitHub Actions' structure, syntax, and semantics
- Automate processes and implement functionality
- Create your own custom actions with Docker, JavaScript, or shell approaches
- Troubleshoot and debug workflows that use actions
- Combine actions with GitHub APIs and other integration options
- Identify ways to securely implement workflows with GitHub Actions
- Understand how GitHub Actions compares to other options

[^ Read less](#)

@techupskills

techupskills.com | techskillstransformations.com



© 2023 Brent C. Laster &  
Tech Skills Transformations LLC

# Review

# Remote vs Local Branches

## User 1

```
$ git clone ...
$ git checkout features
$ git status
```

On branch features  
Your branch is up to date with 'origin/features'.  
**<edit file(s)>**

```
$ git commit -am "update"
```

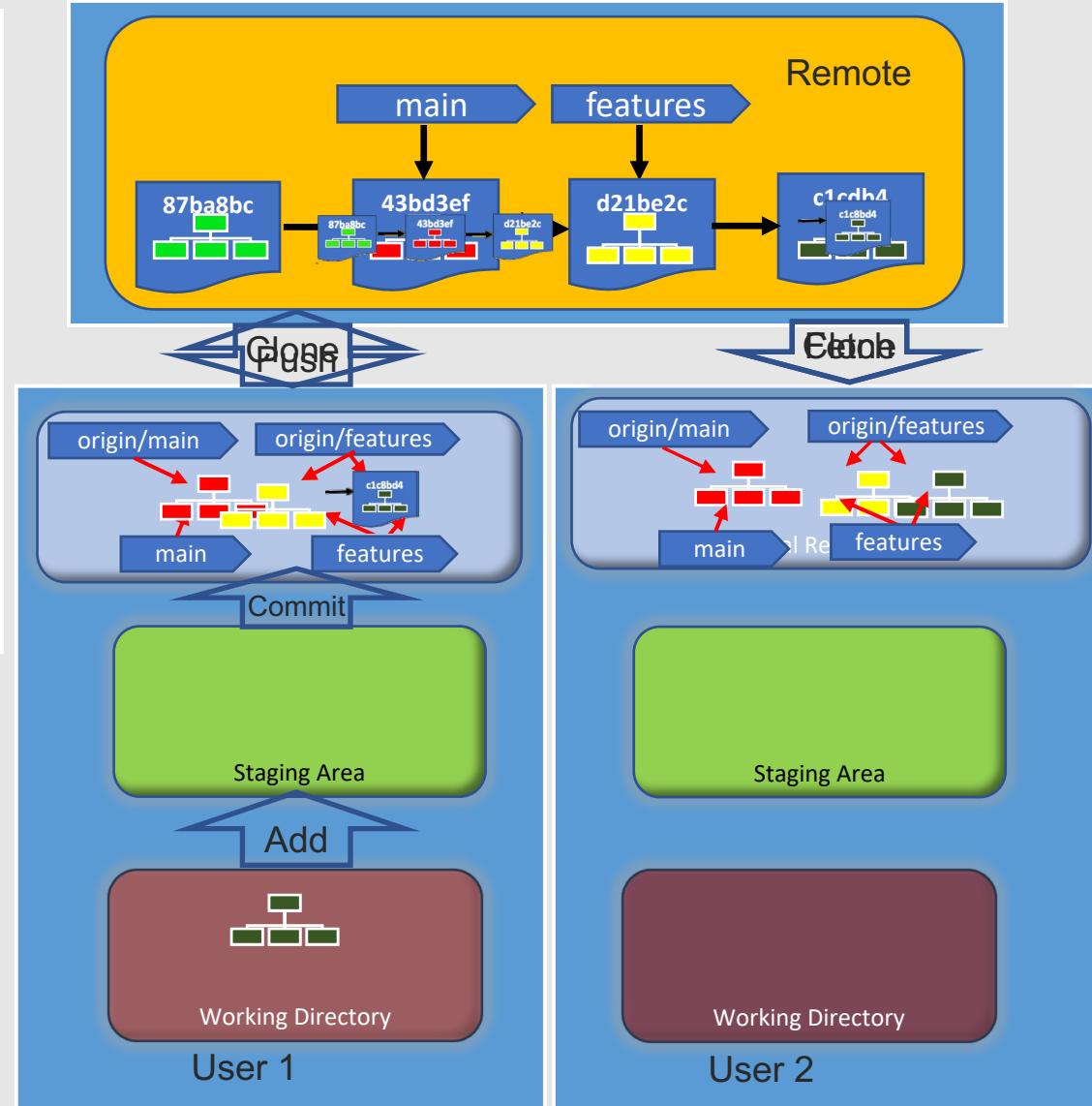
```
$ git status
```

On branch feature  
Your branch is ahead of 'origin/features' by 1 commit.  
(use "git push" to publish your local commits)  
**\$ git push**

## User 2

```
$ git clone ...
$ git checkout features
$ git fetch
$ git status
```

Your branch is behind 'origin/features' by 1 commit, and can be fast-forwarded.  
(use "git pull" to update your local branch)  
**\$ git pull OR \$ git merge origin/features**





# Switching between Branches

**Command: git checkout <branch>**

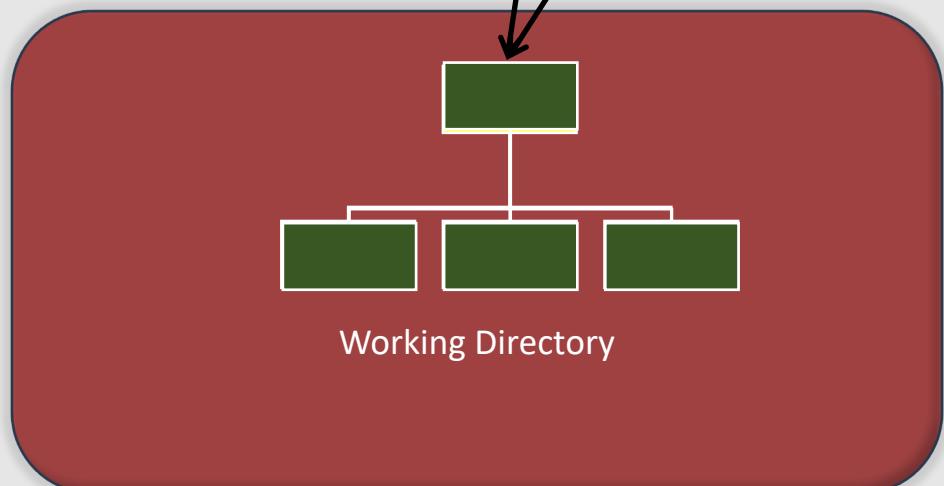
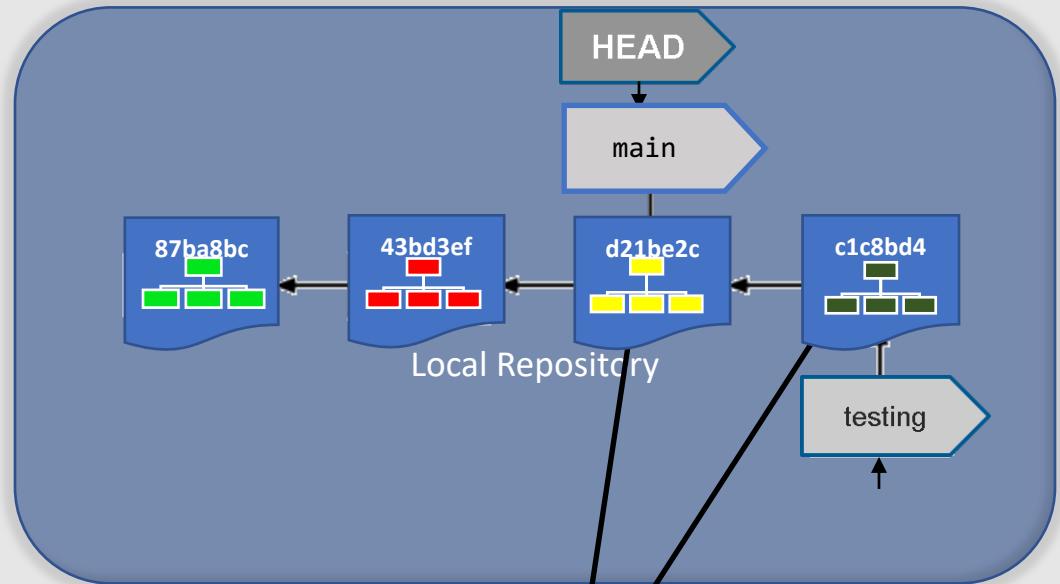
`git checkout main`

- Does three things
  - Moves HEAD pointer back to <branch>
  - Reverts files in working directory to snapshot pointed to by <branch>
  - Updates indicators

`git checkout testing`

`git checkout main`

`git checkout testing`



## New Content



# Command: Worktrees

- Purpose - Allows multiple, separate Working Areas attached to one Local Repository
- Use case - Simultaneous development in multiple branches
- Syntax

```
git worktree add [-f] [--detach] [--checkout] [--lock] [-b <new-branch>]
<path> [<commit-ish>]
git worktree list [--porcelain]
git worktree lock [--reason <string>] <worktree>
git worktree move <worktree> <new-path>
git worktree prune [-n] [-v] [--expire <expire>]
git worktree remove [-f] <worktree>
git worktree unlock <worktree>
```

- Notes
  - “Traditional” working directory is called the *main working tree*; Any new trees you create with this command are called *linked working trees*
  - Information about working trees is stored in the .git area (assuming .git default GIT\_DIR is used)
  - Working tree information is stored in .git/worktrees/<name of worktree>.



# Worktrees - syntax and usage

- Syntax

- *git worktree add [-f] [--detach] [--checkout] [--lock] [-b <new-branch>] <path> [<branch>]*
- *git worktree list [--porcelain]*
- *git worktree prune [-n] [-v] [--expire <expire>]*

- Subcommands

- Add - create a separate working tree for specified branch
  - » has checked out copy of the branch
  - » if no branch specified, uses same name as temp area
- List - lists out current set of working trees active for this repo
  - » porcelain - consistent and backwards-compatible format
- Prune - removes worktree information from the .git area
  - » only applies after worktree directory tree has been removed

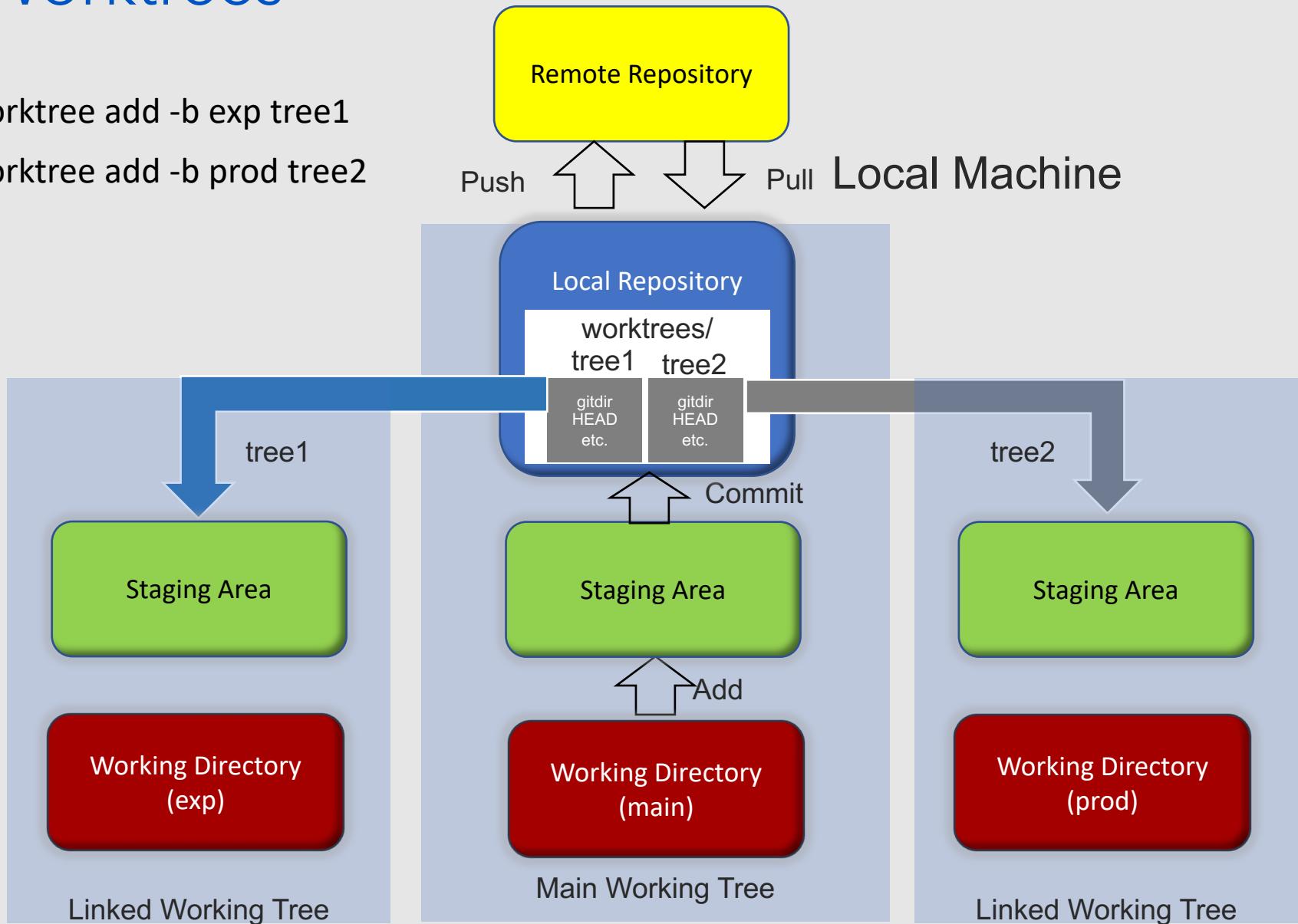


# Worktrees - meta-information

- Information about working trees is stored in the .git area (assuming .git default GIT\_DIR is used)
- Working tree information is stored in .git/worktrees/<name of worktree>.
- Working trees can be created on removable media
  - To persist after unmounting use “lock” subcommand or option on add
  - Use “--reason” to specify a reason for the lock if using lock subcommand



- git worktree add -b exp tree1
- git worktree add -b prod tree2

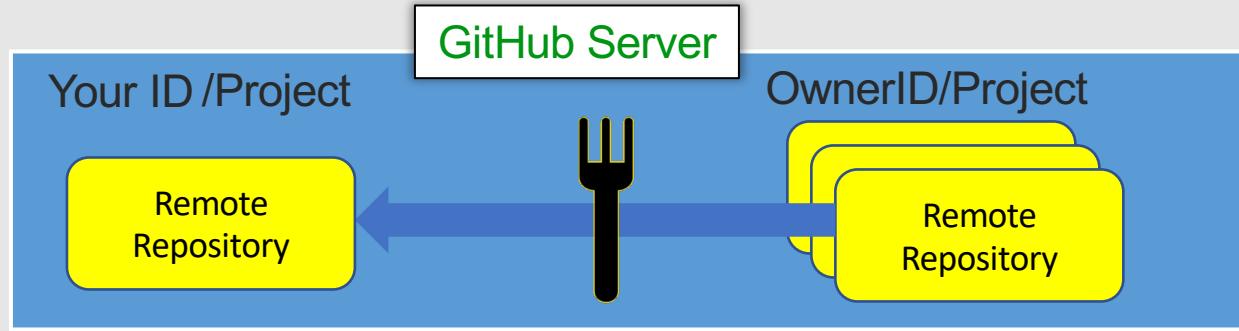


## Lab 11 - Working with Worktrees

Purpose: In this lab, we'll get some experience with how to work on multiple branches at the same time



# Forking Projects



1. User creates account on github.
2. User “forks” an existing github repository.

github.com/skillrepos

github.com/techupskills/calc3

Search or jump to... Search or jump to... Pull requests Issues Marketplace Explore

skillrepos / calc3 forked from skillrepos/calc3

Star 0 Fork 0

Code Issues Actions Projects Wiki Security Insights Settings

main 2 branches 0 tags

This branch is 7 commits ahead of skillrepos:main.

Brent Laster Merge branch 'main' of ssh://github.com/techupskills/calc3 into main

README.md Update README.md

calc.html add avg function

Clone HTTPS SSH GitHub CLI

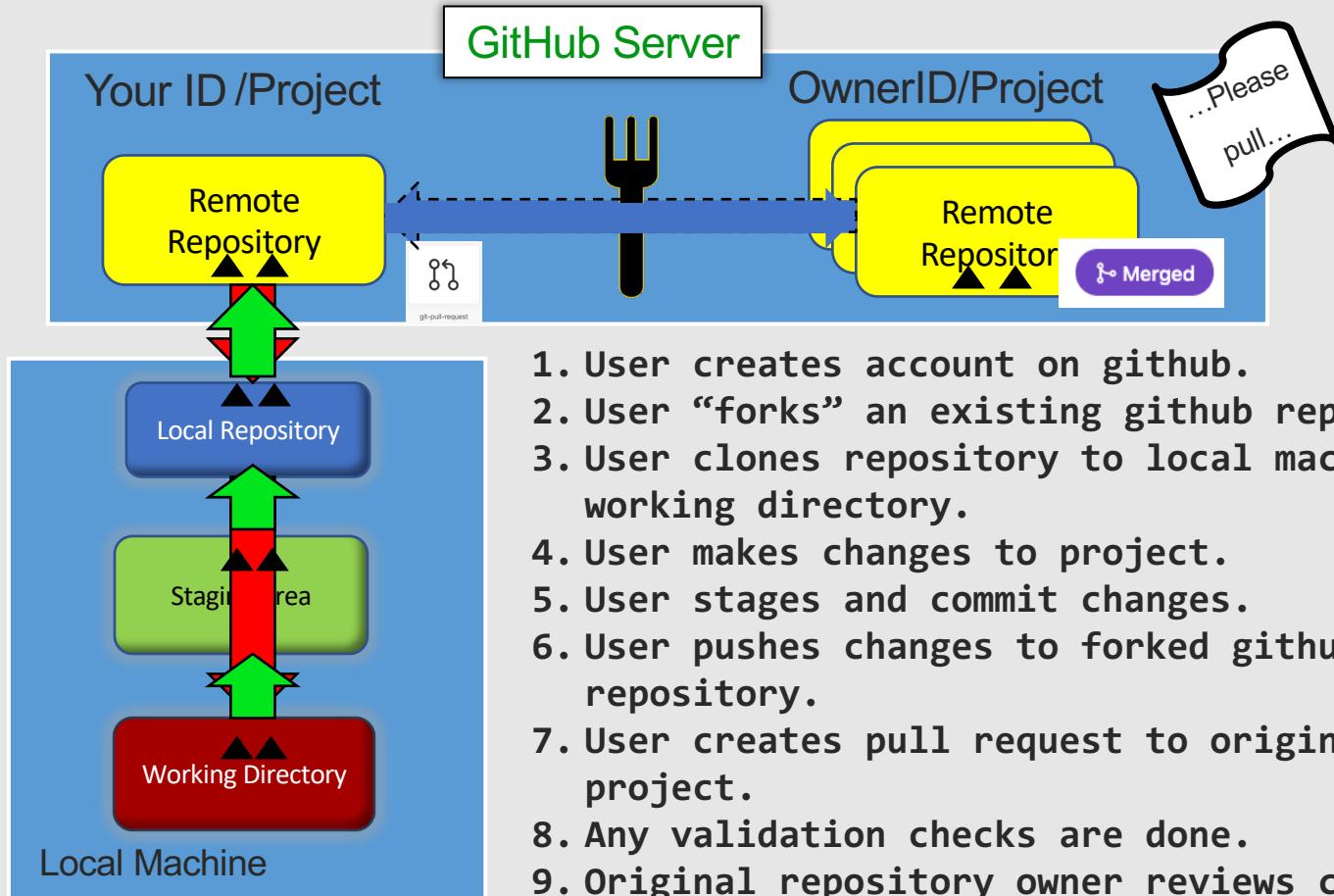
<https://github.com/techupskills/calc3>

Use Git or checkout with SVN using the web URL...

Open with GitHub Desktop



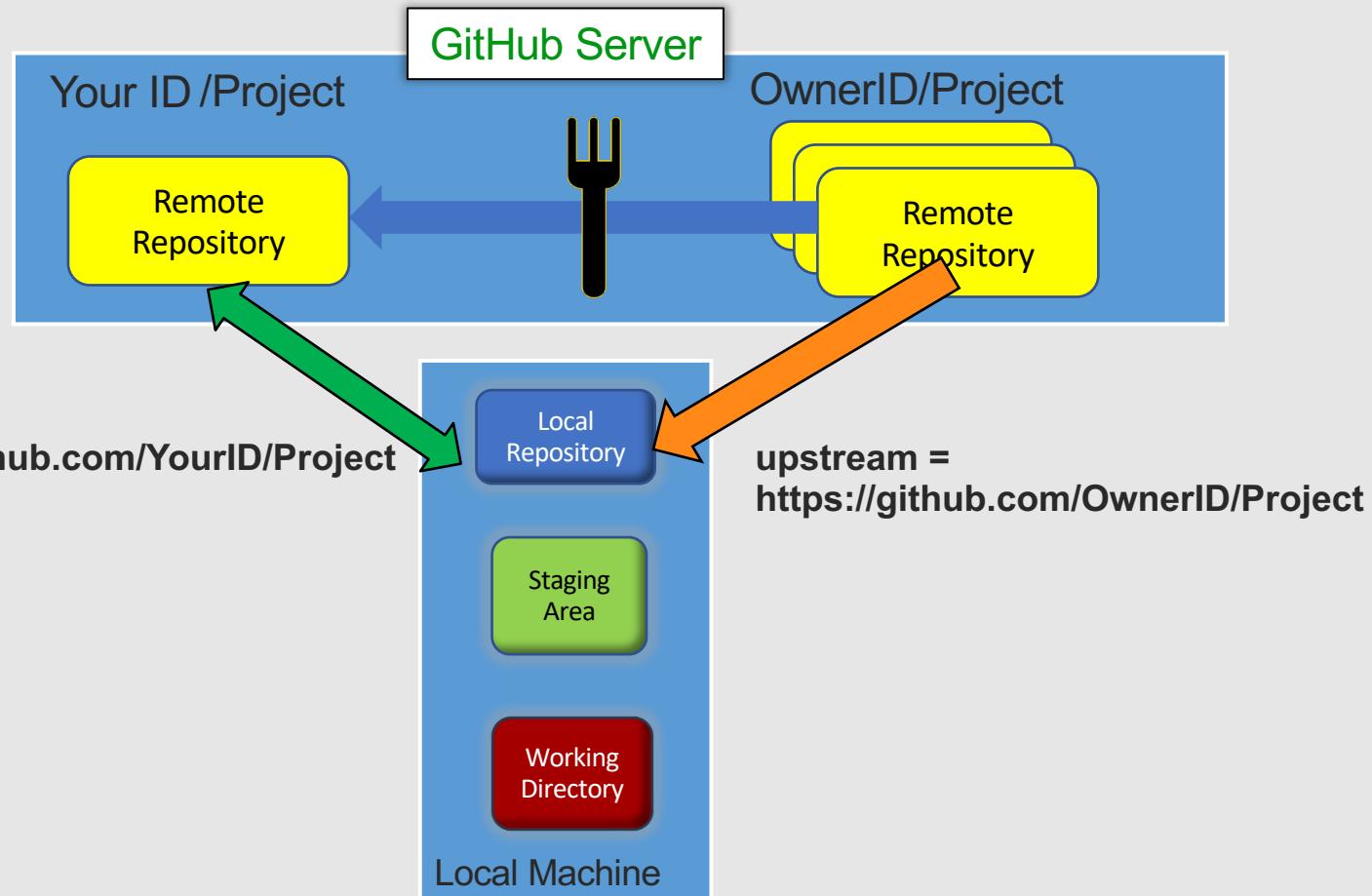
# Pull Requests From Forked Projects



1. User creates account on github.
2. User “forks” an existing github repository.
3. User clones repository to local machine and working directory.
4. User makes changes to project.
5. User stages and commit changes.
6. User pushes changes to forked github repository.
7. User creates pull request to original project.
8. Any validation checks are done.
9. Original repository owner reviews changes.
10. If accepted, original repository owner merges changes into their repository.
11. Pull request is closed.



# Side note: Multiple Remotes





# PR Prompt on Command Line

- When push is done to new branch...

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

remote:

remote: Create a pull request for 'test' on GitHub by visiting:

remote: [https://github.com/<your github userid>/super\\_calc/pull/new/test](https://github.com/<your github userid>/super_calc/pull/new/test)

remote:

To [https://github.com/<your github userid>/super\\_calc.git](https://github.com/<your github userid>/super_calc.git)

\* [new branch] test -> test



# Pull Requests - Choosing what to compare

- Can compare/open PR for branches in same project

The screenshot shows the GitHub interface for the repository `skillrepos / calc3`. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Pull requests' tab is selected. Below the navigation is a section titled 'Comparing changes' with the sub-instruction: 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.' A dropdown menu shows 'base: main' and 'compare: features'. A green checkmark indicates 'Able to merge'. The 'Create pull request' button is visible. The repository stats show 0 comments and 1 contributor.

- Can compare/open PR for different projects (across forks)

The screenshot shows the GitHub interface for the repository `skillrepos / calc3`. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Pull requests' tab is selected. Below the navigation is a section titled 'Comparing changes' with the sub-instruction: 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.' A dropdown menu shows 'base repository: skillrepos/calc3' and 'head repository: techupskills/calc3'. A green checkmark indicates 'Able to merge'. The 'Create pull request' button is visible. The repository stats show 0 comments and 1 contributor.



# Working with Pull Requests

- After push to branch, have opportunity to create PR

tupstudent / super\_calc  
forked from skillrepos/super\_calc

<> Code    Pull requests    Actions    Projects    Wiki    Security    Insights    Settings

dev had recent pushes less than a minute ago

master    2 branches    0 tags

Go to file    Add file    Code

This branch is even with skillrepos:master.

Compare & pull request    Go to file    Add file    Code    Contribute    Fetch upstream

- Select two items for compare/merge

tupstudent / super\_calc  
forked from skillrepos/super\_calc

<> Code    Pull requests    Actions    Projects    Wiki    Security    Insights    Settings

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base repository: tupstudent/super\_calc    base: master    head repository: tupstudent/super\_calc    compare: dev

Choose a Base Repository

Automatically merged.

Filter repos

skillrepos/super\_calc  
brentlaster/super\_calc  
✓ tupstudent/super\_calc

- Create the PR

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master    compare: dev    Able to merge. These branches can be automatically merged.

Dev

Write    Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request



# Pull Request in Progress

- New PR is opened
- Changes available for preview
- Continuous integration run if setup
- Check done for conflicts
- Once all is good can select to “Merge pull request”

https://github.com/tupstudent/super\_calc/pull/1

## Dev #1

**tupstudent** wants to merge 5 commits into `master` from `dev`

Conversation 0    Commits 5    Checks 0    Files changed 1

**tupstudent** commented now

No description provided.

**sasbcl** and others added 5 commits on Sep 4, 2012

- add max function
- add exp function
- add min function
- add avg function
- Updating title

d003b91    482365d    3753e5a    b372aa6    9e4019a

Add more commits by pushing to the `dev` branch on **tupstudent/super\_calc**.

Continuous integration has not been set up  
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch  
Merging can be performed automatically.

Merge pull request    You can also open this in GitHub Desktop or view command line instructions.



# Pull Request Completion

- Confirm merge
- Shows success

Dev #1

**Open** tupstudent wants to merge 5 commits into `master` from `dev`

Conversation 0 Commits 5 Checks 0 Files changed 1

tupstudent commented 3 minutes ago  
No description provided.

sasbcl and others added 5 commits on Sep 4, 2012

- add max function d003b91
- add exp function 482365d
- add min function 3753e5a
- add avg function b372aa6
- Updating title 9e4019a

Add more commits by pushing to the `dev` branch on [tupstudent/super\\_calc](#).

Merge pull request #1 from [tupstudent/dev](#)

Dev

**Confirm merge** **Cancel**

Dev #1

**Merged** tupstudent merged 5 commits into `master` from `dev` now

Conversation 0 Commits 5 Checks 0 Files changed 1

tupstudent commented 5 minutes ago  
No description provided.

sasbcl and others added 5 commits on Sep 4, 2012

- add max function d003b91
- add exp function 482365d
- add min function 3753e5a
- add avg function b372aa6
- Updating title 9e4019a

tupstudent merged commit `c27952c` into `master` now

**Revert**



# PR with merge conflicts

forked from [skillrepos/super\\_calc](#)

Code Pull requests 1 Actions Projects Wiki Security Insights Settings

## add test title #2

[Open](#) techupskills wants to merge 1 commit into `master` from `test`

Conversation 0 Commits 1 Checks 0 Files changed 1

 techupskills commented now

Owner  ...

No description provided.

 add test title 5c6f1ce

Add more commits by pushing to the `test` branch on [techupskills/super\\_calc](#).

 This branch has conflicts that must be resolved

Use the [web editor](#) or the [command line](#) to resolve conflicts.

[Resolve conflicts](#)

**Conflicting files**

calc.html

[Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).



# PR with conflicts - resolve in GitHub

**add test title #2**

Resolving conflicts between `test` and `master` and committing changes → `test`

1 conflicting file	calc.html
 calc.html calc.html	<pre> 1 &lt;html&gt; 2 &lt;head&gt; 3 &lt;&lt;&lt;&lt;&lt; test 4 &lt;title&gt;Test Calc&lt;/title&gt; 5 ===== 6 &lt;title&gt;TechUpskills Calc&lt;/title&gt; 7 &gt;&gt;&gt;&gt;&gt; master 8 9 &lt;script language=javascript type="text/javascript"&gt; 10 11 var plus,minus,divide,multiply,max,pow,min,avg 12 </pre>

**add test title #2**

Resolving conflicts between `test` and `master` and committing changes → `test`

1 conflicting file	calc.html	1 conflict	Prev ^	Next v	Mark as resolved
 calc.html calc.html	<pre> 1 &lt;html&gt; 2 &lt;head&gt; 3 &lt;title&gt;Test Calc&lt;/title&gt; 4 5 6 7 &lt;script language=javascript type="text/javascript"&gt; 8 9 var plus,minus,divide,multiply,max,pow,min,avg </pre>				Mark as resolved

**add test title #2**

Resolving conflicts between `test` and `master` and committing changes → `test`

1 conflicting file	calc.html	✓ Resolved
 calc.html calc.html	<pre> 1 &lt;html&gt; 2 &lt;head&gt; 3 4 &lt;title&gt;Test Calc&lt;/title&gt; 5 6 7 &lt;script language=javascript type="text/javascript"&gt; 8 9 var plus,minus,divide,multiply,max,pow,min,avg </pre>	Commit merge



# PR with conflicts - resolve with command line

The screenshot shows a GitHub pull request page for a repository named `tupstudent/super_calc`. The pull request has been updated with a new commit titled "Updating title". A message indicates that the branch has conflicts that must be resolved. A red oval highlights the link to the "command line" for resolving conflicts. Another red oval highlights the detailed steps for performing a manual merge via the command line.

Updating title #2  
tupstudent wants to merge 1 commit into `master` from `test`

-o Updating title b486c38

Add more commits by pushing to the `test` branch on `tupstudent/super_calc`.

This branch has conflicts that must be resolved  
Use the [web editor](#) or the [command line](#) to resolve conflicts. [Resolve conflicts](#)

**Conflicting files**  
`calc.html`

[Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

**Checkout via command line**  
If you cannot merge a pull request automatically here, you have the option of checking it out via command line to resolve conflicts and perform a manual merge.

[HTTPS](#) [Git](#) [Patch](#) [https://github.com/tupstudent/super\\_calc.git](https://github.com/tupstudent/super_calc.git)

Step 1: From your project repository, bring in the changes and test.

```
git fetch origin
git checkout -b test origin/test
git merge master
```

Step 2: Merge the changes and update on GitHub.

```
git checkout master
git merge --no-ff test
git push origin master
```



# GitHub Project Settings for PRs

https://github.com/tupstudent/super\_calc/settings

...

Merge button

When merging pull requests, you can allow any combination of merge commits, squashing, or rebasing. At least one option must be enabled. If you have linear history requirement enabled on any protected branch, you must enable squashing or rebasing.

**Allow merge commits**  
Add all commits from the head branch to the base branch with a merge commit.

**Allow squash merging**  
Combine all commits from the head branch into a single commit in the base branch.

**Allow rebase merging**  
Add all commits from the head branch onto the base branch individually.

You can allow setting pull requests to merge automatically once all required reviews and status checks have passed.

**Allow auto-merge**  
Waits for merge requirements to be met and then merges automatically. [Learn more](#)

After pull requests are merged, you can have head branches deleted automatically.

**Automatically delete head branches**  
Deleted branches will still be able to be restored.

## Lab 12 - Working with Pull Requests

Purpose: In this lab, we'll see how to create and merge Pull Requests within a GitHub repository

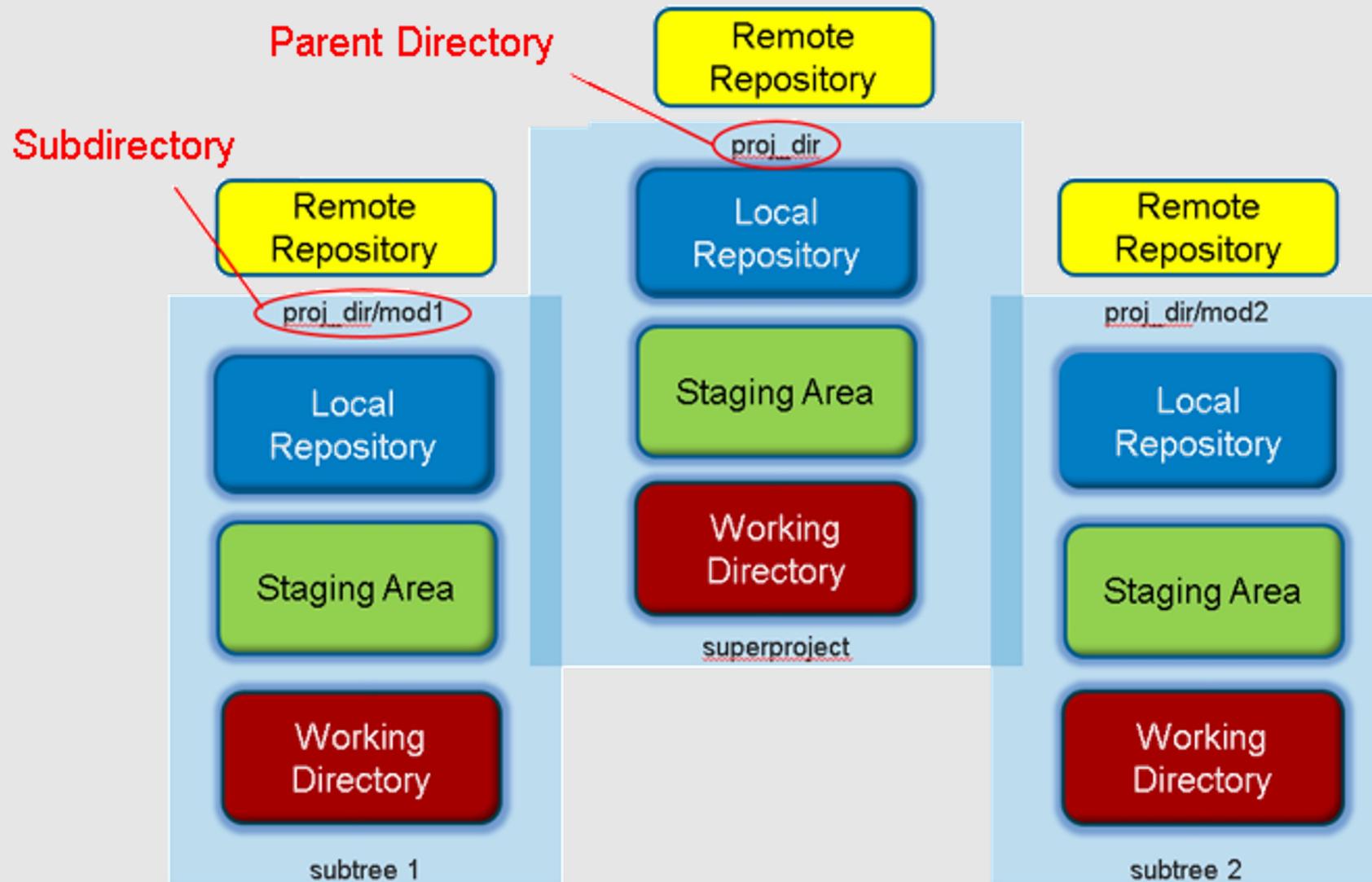


# Command: Subtrees

- Purpose - Allows including a **copy** of a separate repository with your current repository
- Use case - include a **copy** of a Git repository for one or more dependencies along with the original repository for a project
- Syntax  
`git subtree add -P <prefix> <commit>`  
`git subtree add -P <prefix> <repository> <ref>`  
`git subtree pull -P <prefix> <repository> <ref>`  
`git subtree push -P <prefix> <repository> <ref>`  
`git subtree merge -P <prefix> <commit>`  
`git subtree split -P <prefix> [OPTIONS] [<commit>]`
- Notes
  - No links like a submodule - just a copy in a subdirectory
  - Advantage - no links to maintain
  - Disadvantage - extra content to carry around with your



# Subtrees - overall view





# Subtrees - adding a project as a subtree

32

- Simplest form, specify: prefix, remote path to repository, branch (optional)
- Suppose we clone down a project - myproject
- And, on the remote side, we have another project, subproj.
- Now, we can add subproj as a subtree of myproject (--prefix subproject).
- Subtree now shows subproject in structure and history

```
~/subtrees/local/myproject$ ls
file1.txt  file2.txt  file3.txt  subproject/
~/subtrees/local/myproject$ ls subproject/
subfile1.txt  subfile2.txt
```

```
$ git clone ../remotes/myproj.git myproject
Cloning into 'myproject'...
done.
```

```
~/subtrees/remotes$ ls -la subproj.git
total 32
drwxr-xr-x  11 dev  staff   374B Aug  2 20:58 .
drwxr-xr-x   4 dev  staff   136B Aug  2 20:59 ..
-rw-r--r--   1 dev  staff    23B Aug  2 20:58 HEAD
drwxr-xr-x   2 dev  staff    68B Aug  2 20:58 branches/
-rw-r--r--   1 dev  staff   164B Aug  2 20:58 config
-rw-r--r--   1 dev  staff    73B Aug  2 20:58 description
drwxr-xr-x  11 dev  staff   374B Aug  2 20:58 hooks/
drwxr-xr-x   3 dev  staff   102B Aug  2 20:58 info/
drwxr-xr-x   9 dev  staff   306B Aug  2 20:58 objects/
-rw-r--r--   1 dev  staff    98B Aug  2 20:58 packed-refs
drwxr-xr-x   4 dev  staff   136B Aug  2 20:58 refs/
```

```
~/subtrees/local$ cd myproject
~/subtrees/local/myproject$ git subtree add --prefix \
subproject ~/subtrees/remotes/subproj.git master
git fetch /Users/dev/subtrees/remotes/subproj.git master
warning: no common commits
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
From /Users/dev/subtrees/remotes/subproj|
 * branch                  master      -> FETCH_HEAD
Added dir 'subproject'
```

```
~/subtrees/local/myproject$ git log --oneline
7d4f436 Add 'subproject/' from commit '906b5234f366bb2a419953a1edfb590aadc32263'
906b523 Add subfile2
5f7a7db Add subfile1
fada8bb Add file3
ef21780 Add file2
73e59ba Add file1
```



# Subtrees - tips for adding

- Can create a remote to simplify pointing to remote to be added as a subtree

```
~/subtrees/local/myproject$ git remote add sub_origin ~/subtrees/remotes/subproj.git
```

- Use “squash” option to compress history from remote before adding it

```
~/subtrees/local/myproject$ git subtree add --prefix subproject --squash \
sub_origin master
```

- History will now show “squashed” reference in record for history

```
$ git log --oneline
6b109f0 Merge commit 'f7c3147d6df0609745228cc5083bb6c7d0b07d1a' as 'subproject'
f7c3147 Squashed 'subproject/' content from commit 906b523
fada8bb Add file2
ef21780 Add file2
73e59ba Add file1
dev@defaults-MacBook-Pro:~/subtrees/local/myproject$ git log -2
commit 6b109f0d5540642218d442297569b498f8e12396
Merge: fada8bb f7c3147
Author: Brent Laster <bl2@nclasters.org>
Date:   Tue Aug 2 21:15:06 2016 -0400

Merge commit 'f7c3147d6df0609745228cc5083bb6c7d0b07d1a' as 'subproject'

commit f7c3147d6df0609745228cc5083bb6c7d0b07d1a
Author: Brent Laster <bl2@nclasters.org>
Date:   Tue Aug 2 21:15:06 2016 -0400

Squashed 'subproject/' content from commit 906b523

git-subtree-dir: subproject
git-subtree-split: 906b5234f366bb2a419953aledfb590aadcd32263
```



# Subtrees - updating and merging

- Update command - similar to add

```
$ git subtree pull --prefix subproject sub_origin master --squash
```

- pulls down the latest content from the remote into the subtree area
- --squash compresses the history again
  - can be omitted, but usually simplifies things
- also a *git subtree merge* command to merge commits up to a desired point into a subproject denoted by the --prefix argument
  - can be used to merge local changes to a subproject, while git subtree pull reaches out to the remote to get changes



# Subtrees vs. subtree merge strategy

- In Git, there is also a merge strategy named *subtree*.
- `git subtree` command (actually a script that's been incorporated into Git) is not the same thing as the subtree merge strategy.
- Git chooses the best strategy depending on the situation.
- The subtree merge strategy is designed to be used when two trees are being merged and one is a subtree (subdirectory) of the other.
- Subtree merge strategy tries to shift the subtrees to be at the same level in order to merge similar structures.
- For more information, search for *subtree* in the help page for `merge`.

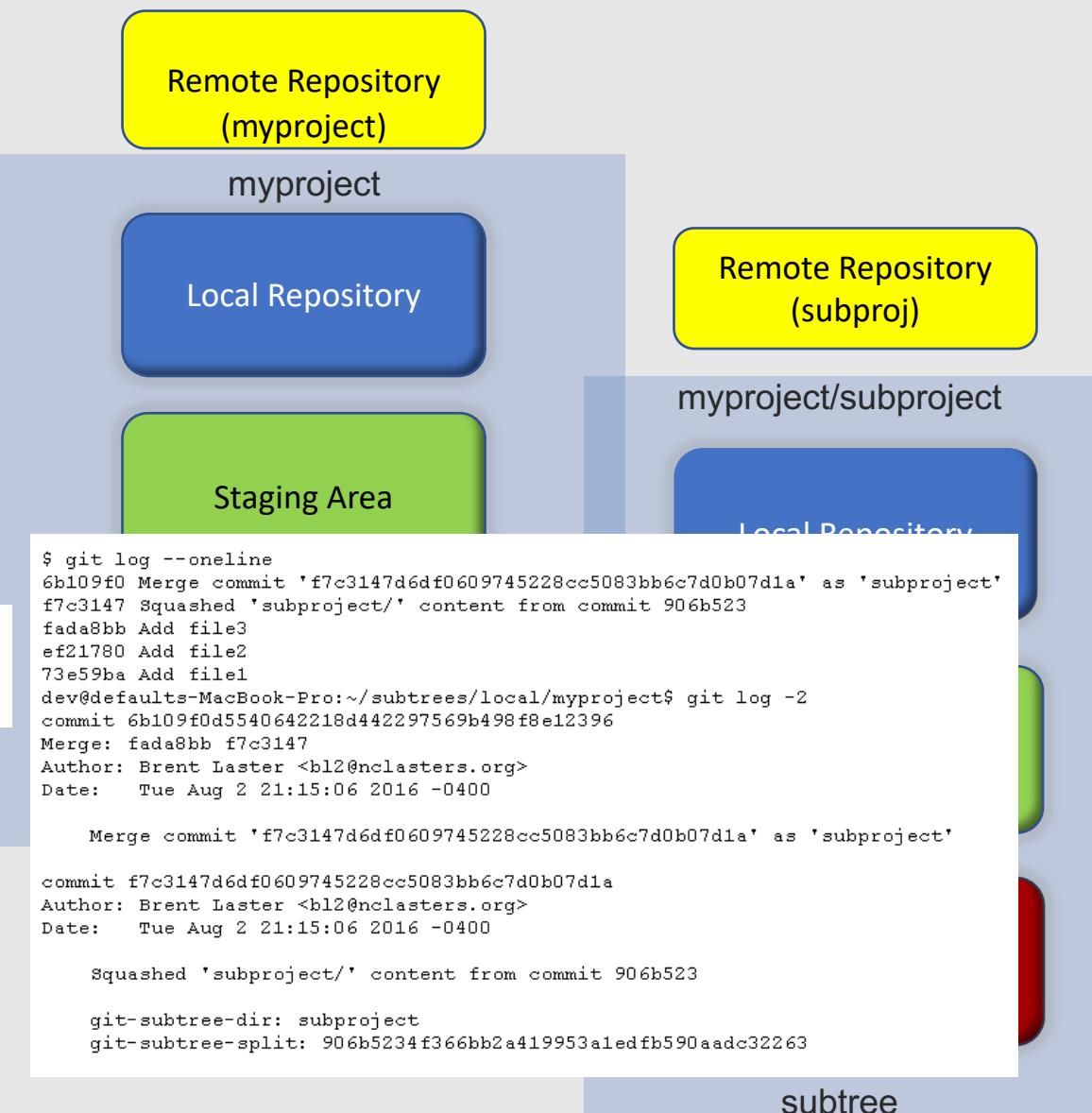


# Subtrees

- Adding a copy from a remote as a subtree
- cd myproject
- git subtree add --prefix subproject --squash subproj.git main
  - Use “prefix” option to specify path for subproject
  - Use “squash” option to compress history from remote before adding it
  - branch (main) is optional
- directory listing of myproject now shows subproj as subdirectory

```
~/subtrees/local/myproject$ ls
file1.txt  file2.txt  file3.txt  subproject/
~/subtrees/local/myproject$ ls subproject
subfile1.txt  subfile2.txt
```

- Looking in the logs of the subproject will show the squashed history
- To get the latest, use pull
- git subtree pull --prefix subproject --squash subproj.git main





# Subtrees - split

- split subcommand can be used to extract a subproject's content into a separate branch
- extracts the content and history related to <prefix> and puts the resulting content at the root of the new branch instead of in a subdirectory

```
~/subtrees/local/myproject$ git subtree split --prefix=subproject \
--branch=split_branch
Created branch 'split_branch'
906b5234f366bb2a419953a1edfb590aadc32263
```

- As output, Git prints out the SHA1 value for the HEAD of the newly created tree
  - Provides a reference to work with for that HEAD if needed
  - New branch shows only the set of content from the subproject that was split out (as opposed to content from the superproject).

```
~/subtrees/local/myproject$ git checkout split_branch
Switched to branch 'split_branch'
~/subtrees/local/myproject$ ls
subfile1.txt  subfile2.txt
~/subtrees/local/myproject$ git log --oneline
906b523 Add subfile2
5f7a7db Add subfile1
```



# Subtree - create new project from split content

- Since can split out content from a subtree, may want to transfer that split content into another project
- Very simple with Git
  - create new, empty project

```
~/subtrees/local/myproject$ cd ~/  
~$ mkdir newproj  
~$ cd newproj  
~/newproj$ git init  
Initialized empty Git repository in /Users/dev/newproj/.git/
```

- pull contents of new branch into the new project (repository)

```
~/newproj$ git pull ~/subtrees/local/myproject split_branch  
remote: Counting objects: 5, done.  
remote: Compressing objects: 100% (3/3), done.  
remote: Total 5 (delta 0), reused 0 (delta 0)  
Unpacking objects: 100% (5/5), done.  
From /Users/dev/subtrees/local/myproject  
 * branch          split_branch -> FETCH_HEAD
```



# Subtree - push

- subtree command also supports a push subcommand
- This command does a split followed by an attempt to push the split content over to the remote
- Example: the following command splits out the subproject directory and then pushes it to the `sub_origin` remote reference and into a new branch named *new branch*:

```
~/subtrees/local/myproject$ git subtree push --prefix=subproject sub_origin new_branch
git push using: sub_origin new_branch
Total 0 (delta 0), reused 0 (delta 0)
To /Users/dev/subtrees/remotes/subproj.git
 * [new branch] 906b5234f366bb2a419953a1edfb590aadcc32263 -> new_branch
~/subtrees/local/myproject$
```

## Lab 13 - Subtrees

Purpose: In this lab, we'll see one way to manage "groups" of repositories in Git



# Hooks – basic concepts

- Definition – a program or script that runs when a certain event happens
- Common uses
  -  Sending email or other notifications when a change is pushed to a repository
  -  Validating that certain conventions have been met before a commit
  -  Appending items to commit messages
  -  Checking the format or existence of certain elements in a commit message
  -  Updating content in the working directory after an operation
  -  Enforcing coding standards



# Hooks - accessing

- By default, hooks come from Git template area and live in .git/hooks
- As of Git 2.9, can have different path to hooks
  - Set by core.hooksPath config value
  - Provides means to have common path to hooks for multiple users or projects (shared hooks)
- After cloning or init, .git/hooks is populated with sample hooks

applypatch-msg.sample	pre-push.sample
commit-msg.sample	pre-rebase.sample
post-update.sample	prepare-commit-msg.sample
pre-applypatch.sample	update.sample
pre-commit.sample	

- Hooks are NOT cloned as part of Git clone
- Can be updated via “git init”



# Hooks - attributes

- Domain

- Local : applypatch-msg, pre-applypatch, post-applypatch, pre-commit, prepare-commit-msg, commit-msg, post-commit, pre-rebase, post-checkout, post-merge, pre-push, post-rewrite, push-to-checkout
- Remote : pre-receive, update, post-receive, post-update, pre-auto-gc

- Return Code

- 0 = success; operation should continue
- Non-0 = not success; operation should abort

- Working Directory Location

- Non-bare repo – working directory root
- Bare repo - repository directory

- Environment Variables

- GIT\_DIR, GIT\_AUTHOR\_DATE, GIT\_AUTHOR\_EMAIL, GIT\_AUTHOR\_NAME



# Hooks - am

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
am	applypatch-msg	pre-applypatch		post-applypatch	

- **am command** – takes a patch (may be from email) applies it, and commits the change
- **applypatch-msg** – operates on commit message for apply part of git am operation (mailing patches)
  - P1 – Name of temporary file with proposed commit message
- **pre-applypatch** – called after patch is applied, but before committed; allows Git to verify patch results look right
- **post-applypatch** – called after patch is applied AND committed; useful for notifications



# Hooks - commit

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
commit	pre-commit	prepare-commit-msg	commit-msg	post-commit	

- **pre-commit** - verify that what's about to be committed meets some condition or criteria and is okay to commit
- **prepare-commit-msg** - **Applypatch-msg** – operates on commit message for apply part of git am operation (mailing patches)
  - P1: Name of temporary file with proposed commit message
  - P2: Type of operation
  - P3: SHA1 value for certain operations
- **commit-msg** – called after patch is applied, but before committed; allows Git to verify patch results look right
  - P1: Name of temporary file with proposed commit message
- **post-commit** – called after patch is applied AND committed; useful for notifications



# Hooks - push

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
push	pre-push	pre-receive	update	post-receive	post-update

- **pre-push** – called prior to a push; can prevent a push
  - P1: Remote reference name
  - P2: Remote URL Extra: STDIN lines of the form <local reference> <local sha1> <remote reference> <remote sha1> LF
- **pre-receive** – runs once for push before updating references
  - No parameters - Extra: STDIN lines of the form <local reference> <local sha1> <remote reference> <remote sha1> LF
- **update** – runs once for each reference to be updated
  - P1: Name of reference being updated
  - P2: Old SHA1 value
  - P3: New SHA1 value
- **post-receive** – runs once after all references are updated; knows old and new
  - No parameters Extra: STDIN lines of the form <old value> <new value> <reference name> LF
- **post-update** – runs after all references; doesn't know old and new
  - P\* (variable number): Name of references being updated



# Hooks - rebase

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
rebase	pre-rebase				

- **pre-rebase** - provides an opportunity to validate that the rebase should go through, issue a warning message, and so on
  - P1: Upstream that the current series of commits come from
  - P2: Branch being rebased (if not the same as P1)
- Example hook – prevents topic branches that have already been merged from being rebased (and so not merged again)



# Hooks – push (to non-bare repository)

48

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
<b>push (to non-bare repo)</b>	<b>push-to-checkout</b>				

- **non-bare repository** – one that has working directory and staging area and checked out branch
- normally push to bare remote repository only, so receive.denyCurrentBranch setting to prevent pushes to non-bare
- **push-to-checkout** – allows to override receive.denyCurrentBranch and sync working directory and staging area to make everything consistent
  - P1: Target commit for updating



# Hooks – commit --amend or rebase

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
commit – amend or rebase				post-rewrite	

- **post-rewrite** – runs after operations that change history (rewrite commits)
  - P1: command that called it
  - <stdin> : <old sha1> <new sha1> [ <optional extra data> ]
- currently no extra data is passed, but might be in future
- runs after commit –amend or rebase (but not filter-branch)
- similar uses to post-checkout and post-merge



# Hooks – gc --auto

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
gc --auto	pre-gc-auto				

- **gc – run garbage collection (cleaning up objects that aren't used anymore)**
  - auto = option to cleanup if there are too many loose objects over a configured threshold
- **hook runs first if --auto option is specified**
  - No parameters
- **can do notification and/or verification before gc**



# Hooks - checkout

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
checkout				post-checkout	

- **post-checkout** – runs after a successful checkout (or after clone unless no-checkout option is used)
  - P1: Previous HEAD (before checkout)
  - P2: Current HEAD (after checkout)
  - P3: Checkout type flag: 1= branch, 2 = file
- can be used to cleanout unwanted files



# Hooks – merge

Git Operation	Pre-operation Hook 1	Pre-operation Hook 2	During-operation Hook	Post-operation Hook 1	Post-operation Hook 2
merge, pull				post-merge	

- **post-merge** – runs after a successful pull or merge (or after clone unless no-checkout option is used)
  - P1: flag that indicates squash or merge
- can be used to do things like set permissions or kick off supporting processes (such as testing)



# A simple post-commit hook

- Intent: Mirror copies of files when committed to local repo if branch starts with “web”.
- Configure hooks.webdir as location to mirror to

```

1 #!/usr/bin/env bash
2
3 echo Running post-commit hook
4
5 web_dir=$(git config hooks.webdir)
6
7 new_head_ref=$(git rev-parse --abbrev-ref HEAD)
8
9 unset GIT_INDEX_FILE
10
11 if [[ -n "$web_dir" ]] && [[ $new_head_ref =~ ^web.*$ ]]; then
12     results=$(git --work-tree="$web_dir" checkout -f)
13 fi

```

path to bash interpreter

simple execution indicator

get value of hooks.webdir  
custom config setting

figure out current HEAD  
(branch)

not needed so unset else  
causes problems for  
checkout to non-Git area

only mirror code if  
conditions are met

if conditions are met, call git and check out  
current code into the configured location

check if web\_dir (config  
value) is set

check if branch starts with  
“web”

using global work-tree option to redirect  
checkout to desired directory

## Lab 14 - Creating a post-commit hook

Purpose: In this lab, we'll see how to put a post-commit hook in place to be active in our local Git environment.

# That's all - thanks!



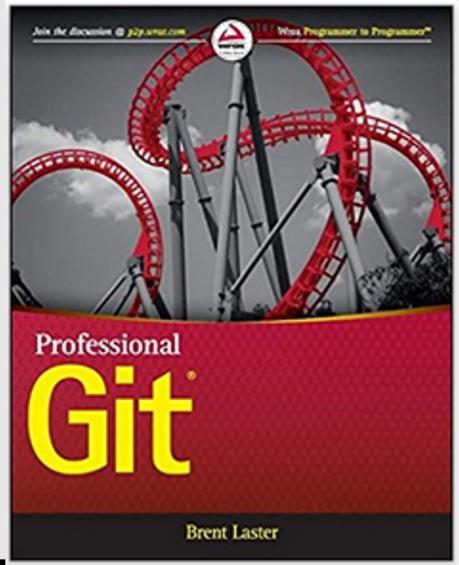
[techskillstransformations.com](http://techskillstransformations.com)  
[getskillsnow.com](http://getskillsnow.com)

## Professional Git 1st Edition

by Brent Laster (Author)

4.5 stars - 7 customer reviews

[Look inside](#) ↓



@techupskills

[techupskills.com](http://techupskills.com) | [techskillstransformations.com](http://techskillstransformations.com)

New courses on Git, K8S, Operators, GitOps and more!

TECH SKILLS TRANSFORMATIONS, LLC

Home Contact Us

TECH LEARNING MADE EASY

Get the instruction you need to upskill now! Click the button below to see upcoming trainings.

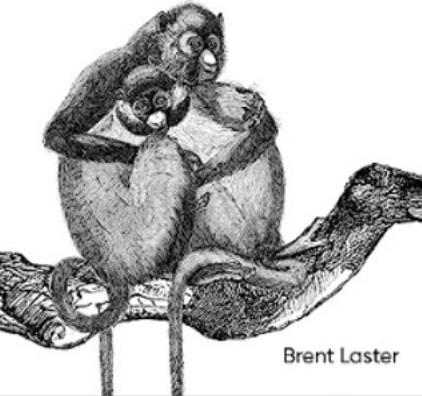
Upcoming live training with O'Reilly Media!



O'REILLY®

# Learning GitHub Actions

Automation and Integration of CI/CD with GitHub



Brent Laster

O'REILLY®



# Jenkins 2 Up & Running

EVOLVE YOUR DEPLOYMENT PIPELINE FOR NEXT GENERATION AUTOMATION

Brent Laster



© 2023 Brent C. Laster &  
Tech Skills Transformations LLC