GIT

version-control system

Outline for today

- The tutorial is for people who have never seen GIT before
  - We won’t be typing in any commands
- Explain the GIT workflow and basic operations visually
- Something for everyone
- Video lectures to learn in more depth!

What is GIT

- Version control
  - Thesis_v1.doc
  - Thesis_v2.doc
  - Thesis_v217.doc
- Keeps track of changes
  - Text changes
  - Most often to manage source code
- Move back and forth between versions
- Compare and see what changed between versions
- Team development and distributing code
- Industry standard

Change Sets (/Snapshots/commits)

- Git stores "change sets"
  - incremental changes applied
  - multiple files
- We'll explore how GIT tracks and merges "change sets"
- Small and simple is usually better
  - Want changes to be focused and independent of each other
  - If 2 change sets make conflicting edits, more difficult to merge
- Usually (but not always) good to have change sets that leave code in a working condition.

The Basics

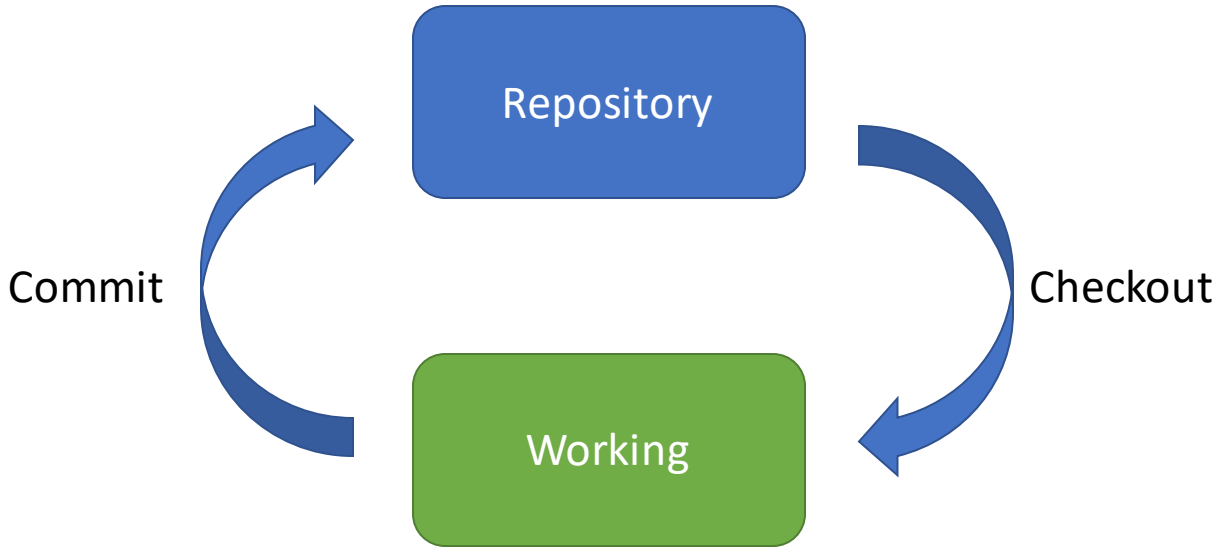
- Create a new folder and navigate into it
- git init command
  - Make this a git repository
  - Track all the files that come and go, changes that are made, and do it for all subfolders too
- .git folder is where it does all storing a tracking
  - This is GIT's workspace, GIT will manage these files
  - No need to mess around unless you know what you're doing

The Basics

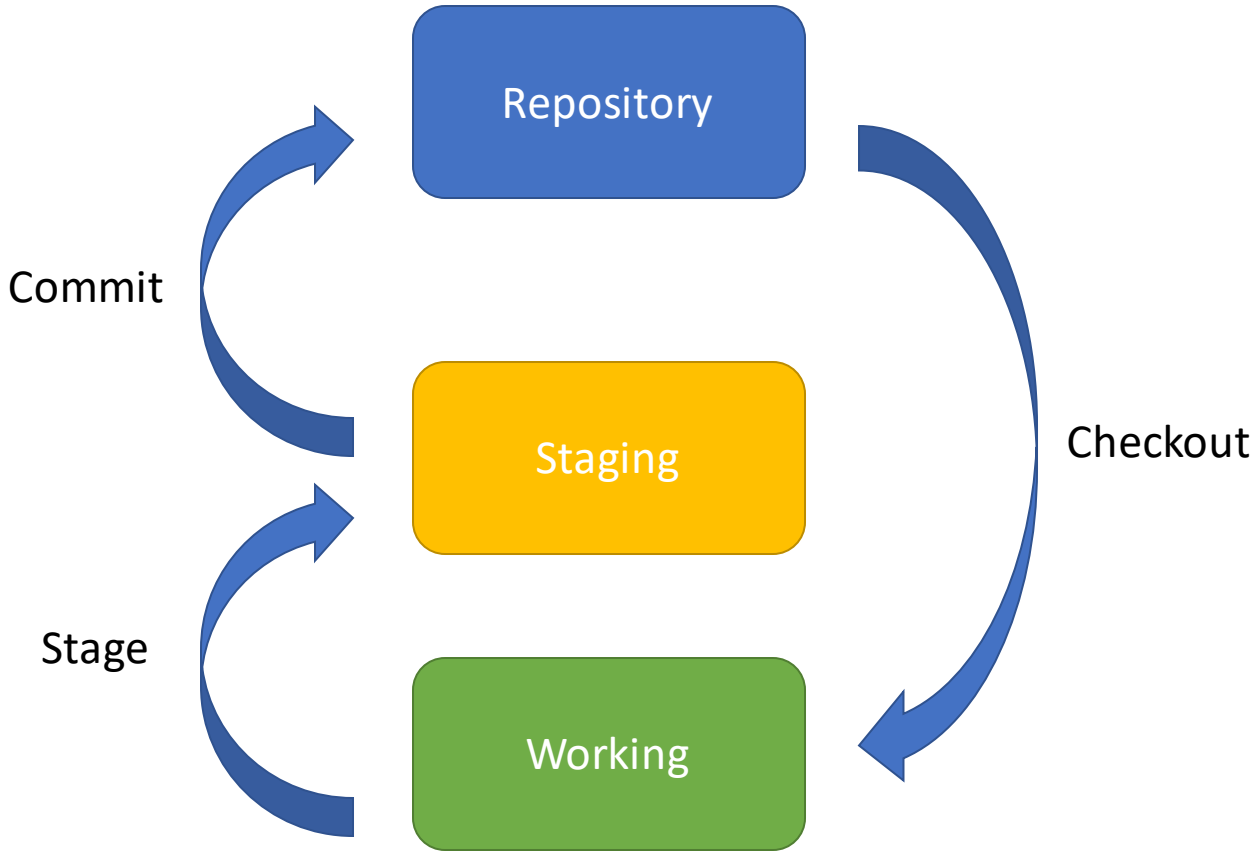
- The basic flow
  - Make changes in **working directory**
  - **Add** change set to **staging directory**
  - **Commit** change set to **repository directory**

3 Tree Architecture

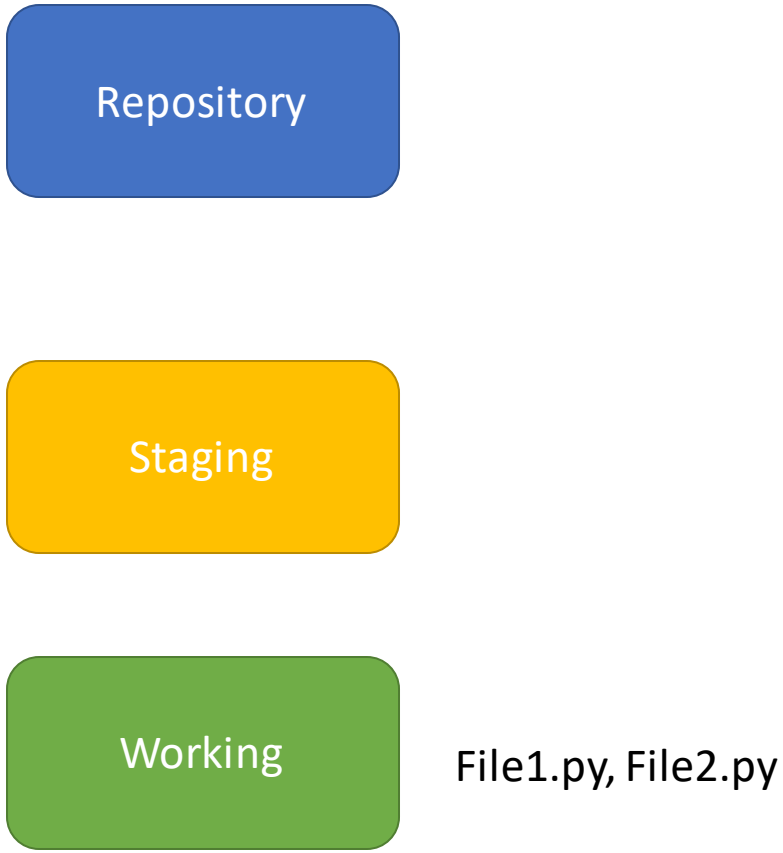
3 Tree Architecture



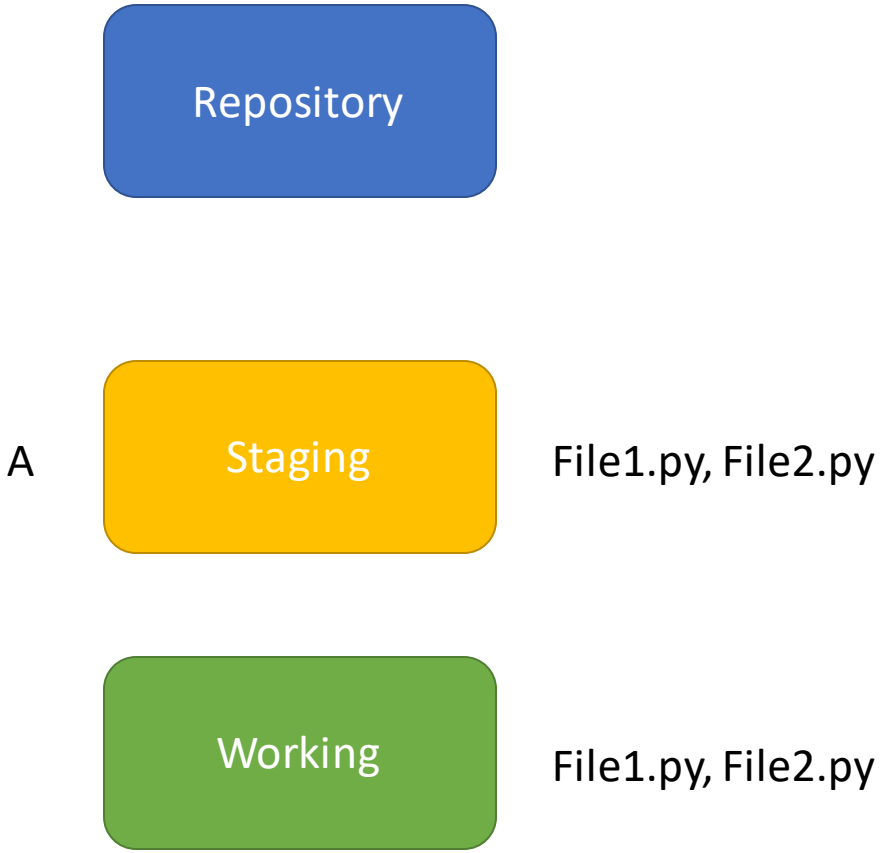
3 Tree Architecture



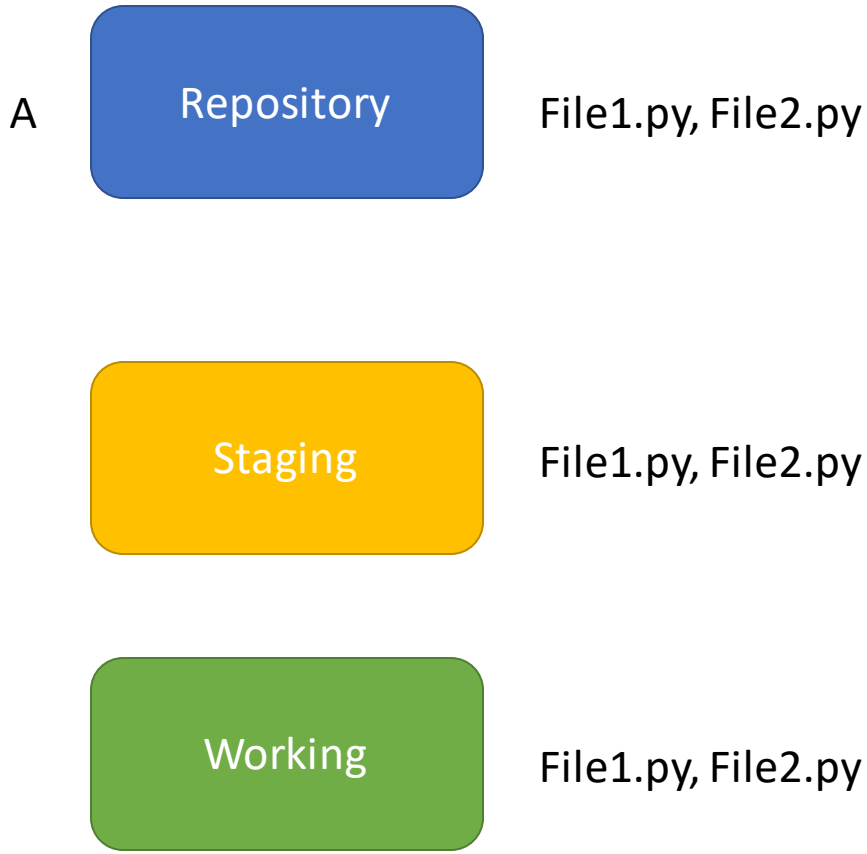
3 Tree Architecture, what's the point of add?



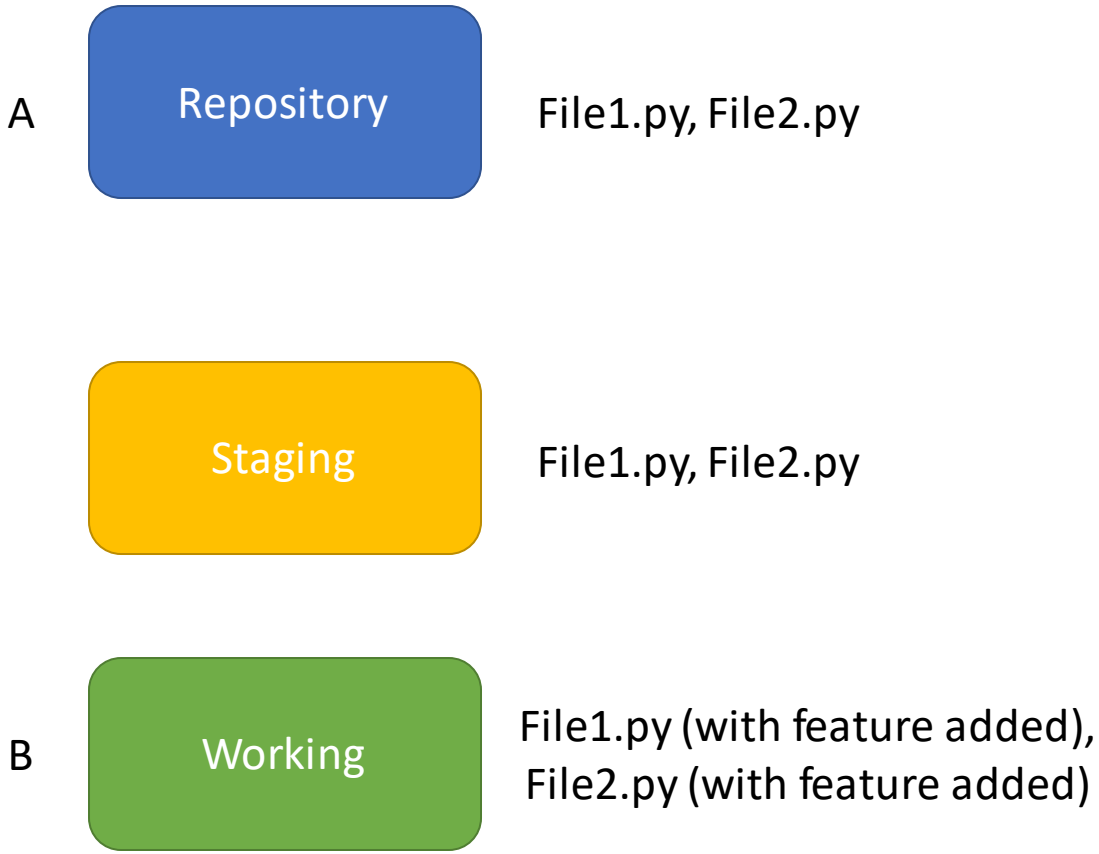
3 Tree Architecture, what's the point of add?



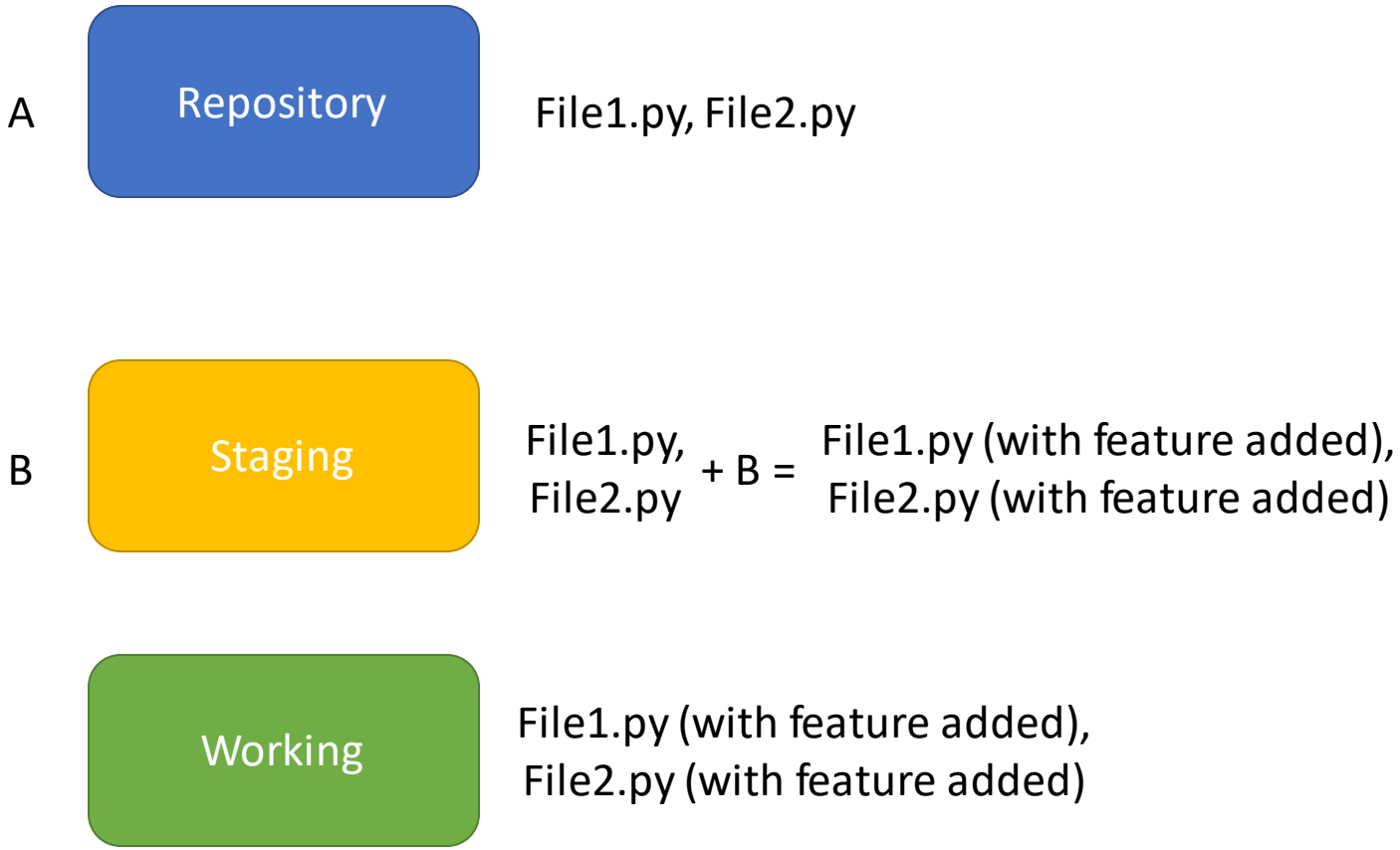
3 Tree Architecture, what's the point of add?



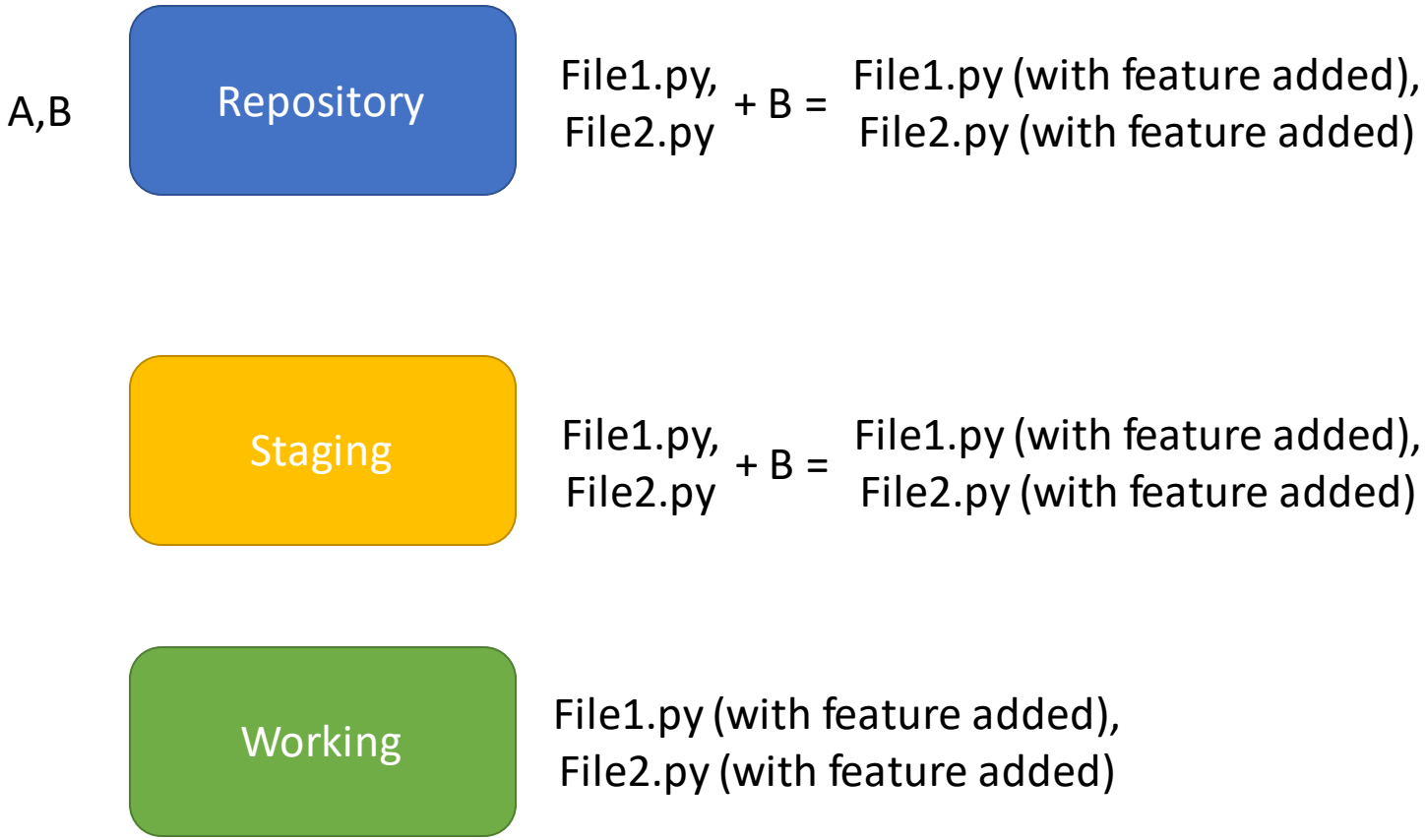
3 Tree Architecture, what's the point of add?



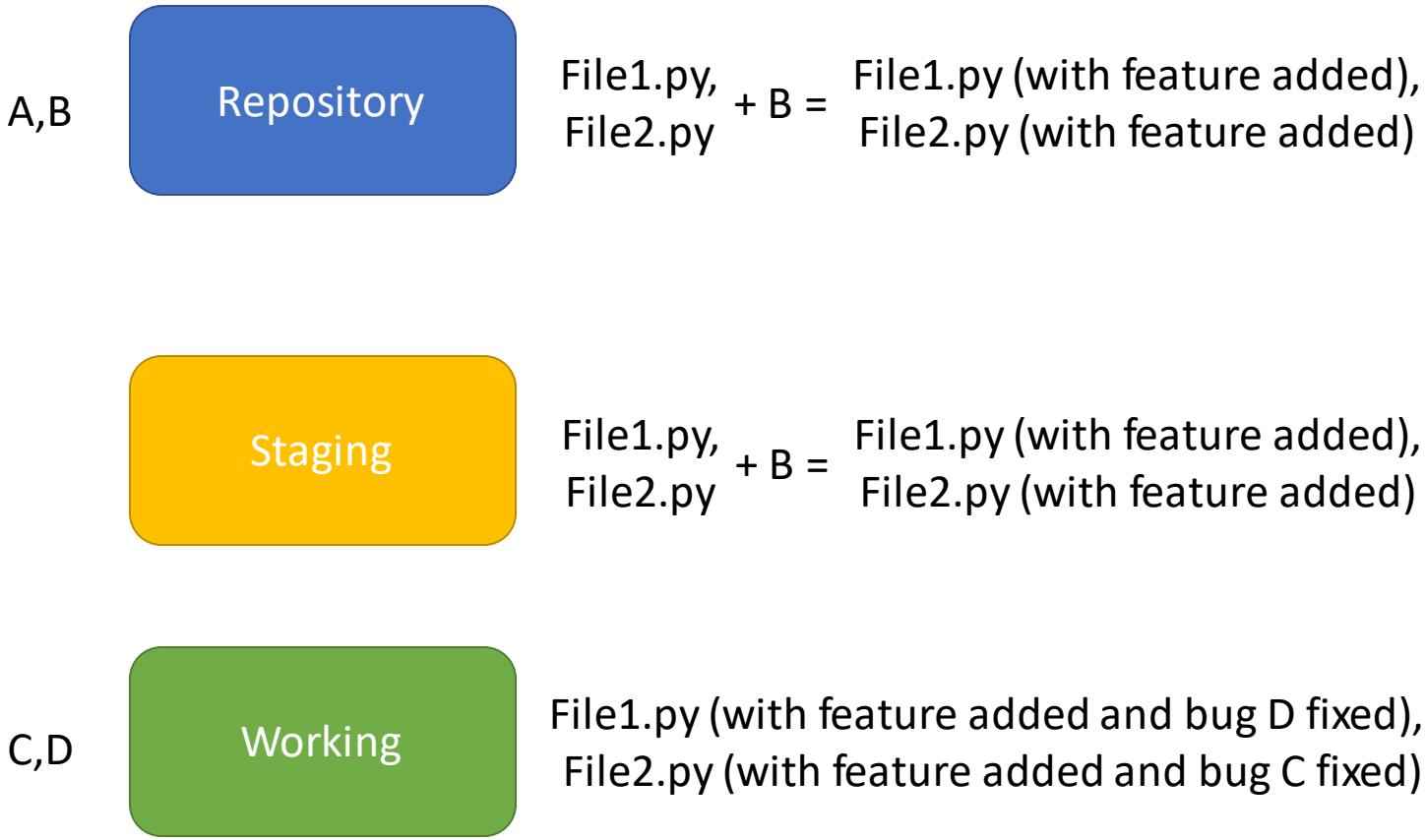
3 Tree Architecture, what's the point of add?



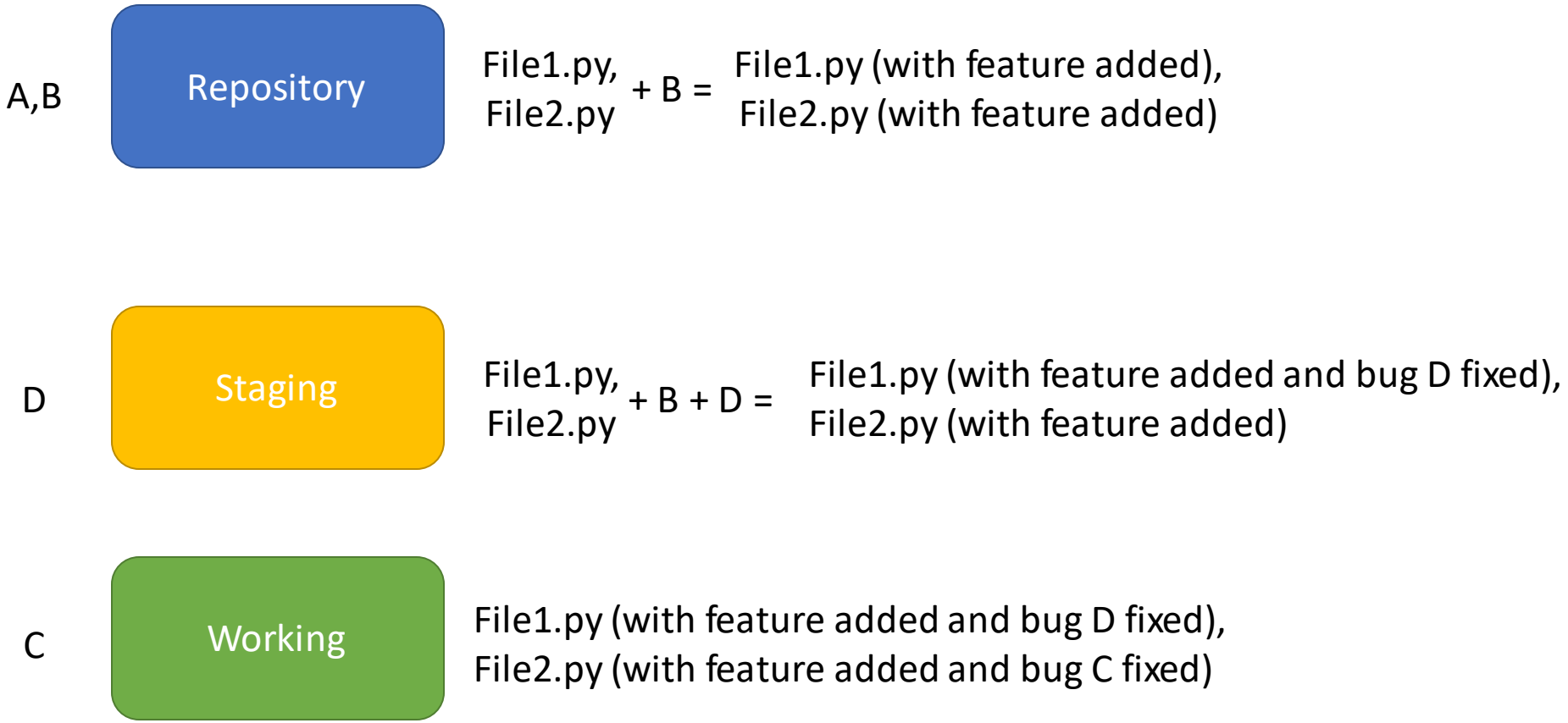
3 Tree Architecture, what's the point of add?



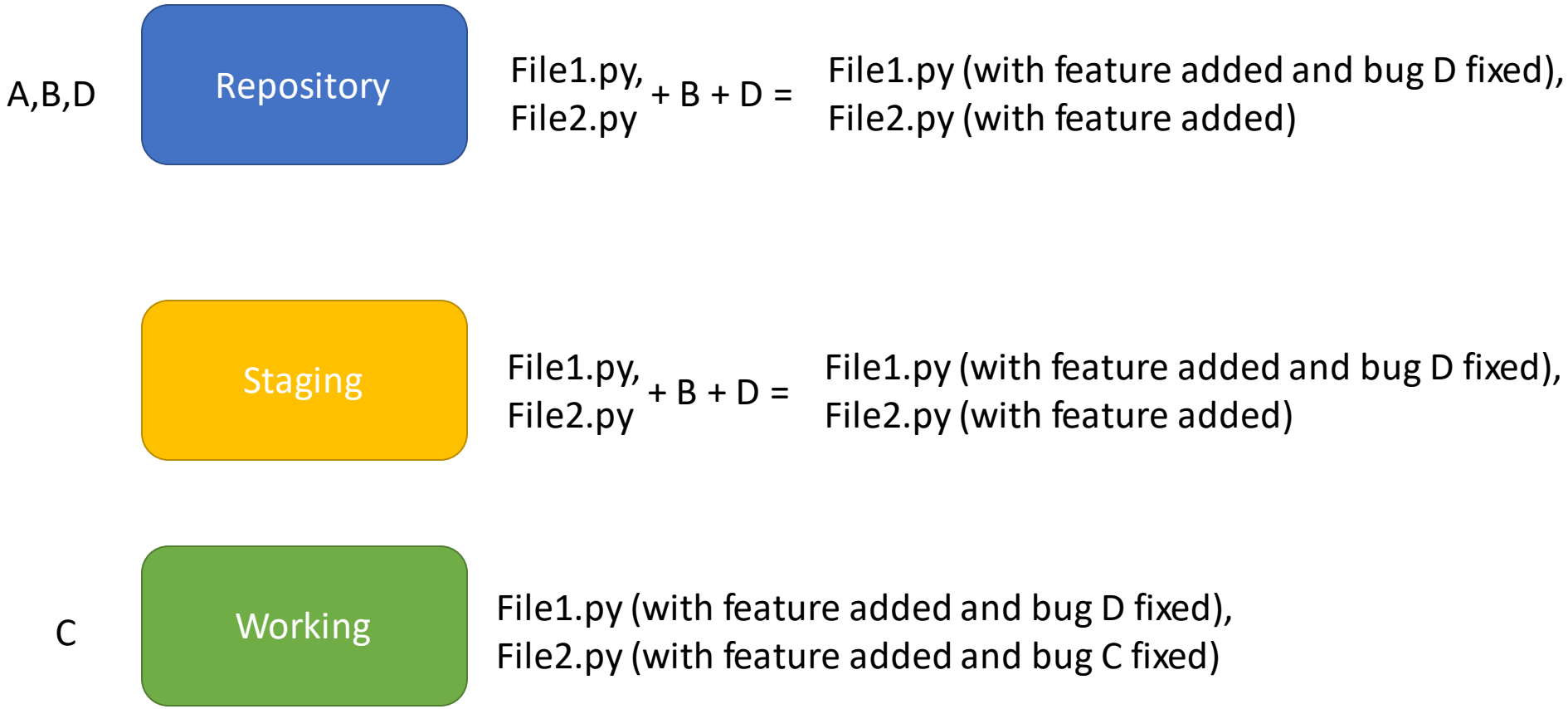
3 Tree Architecture, what's the point of add?



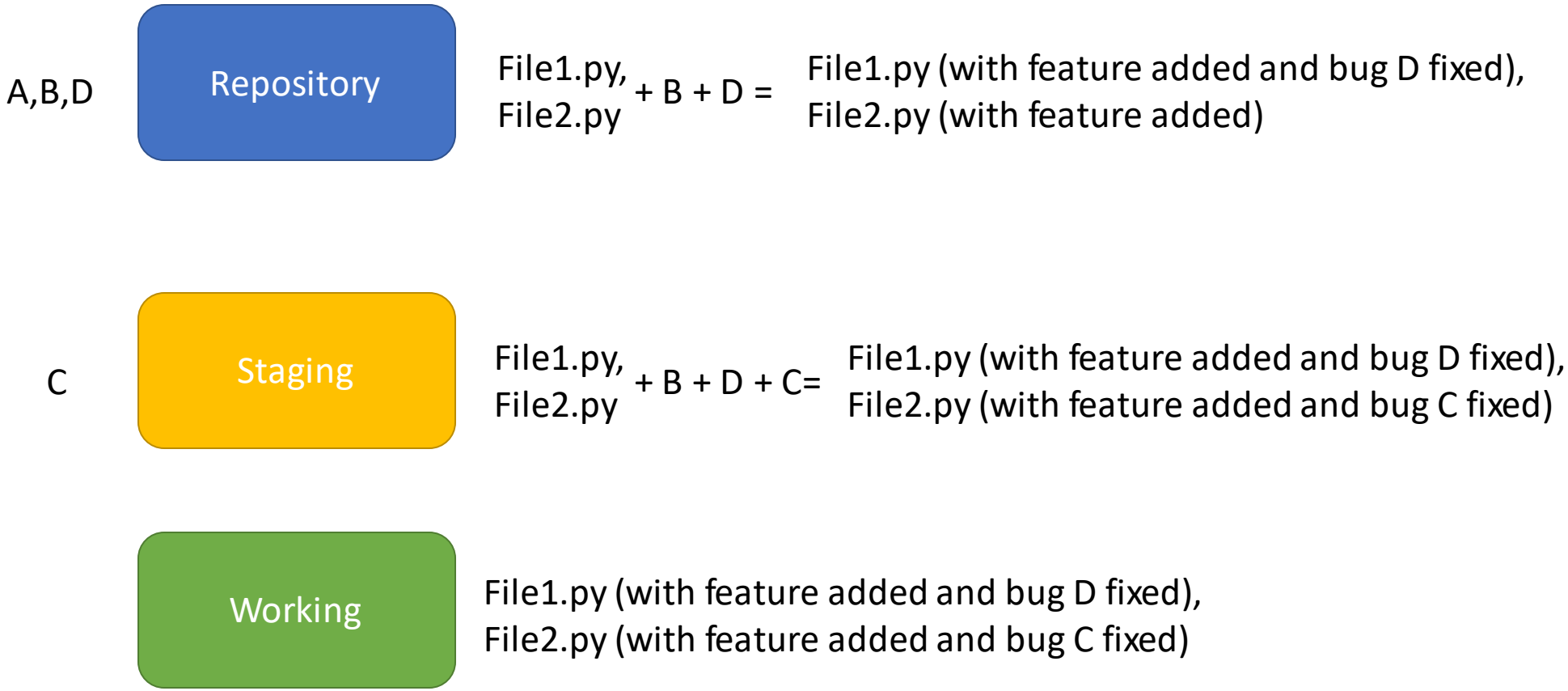
3 Tree Architecture, what's the point of add?



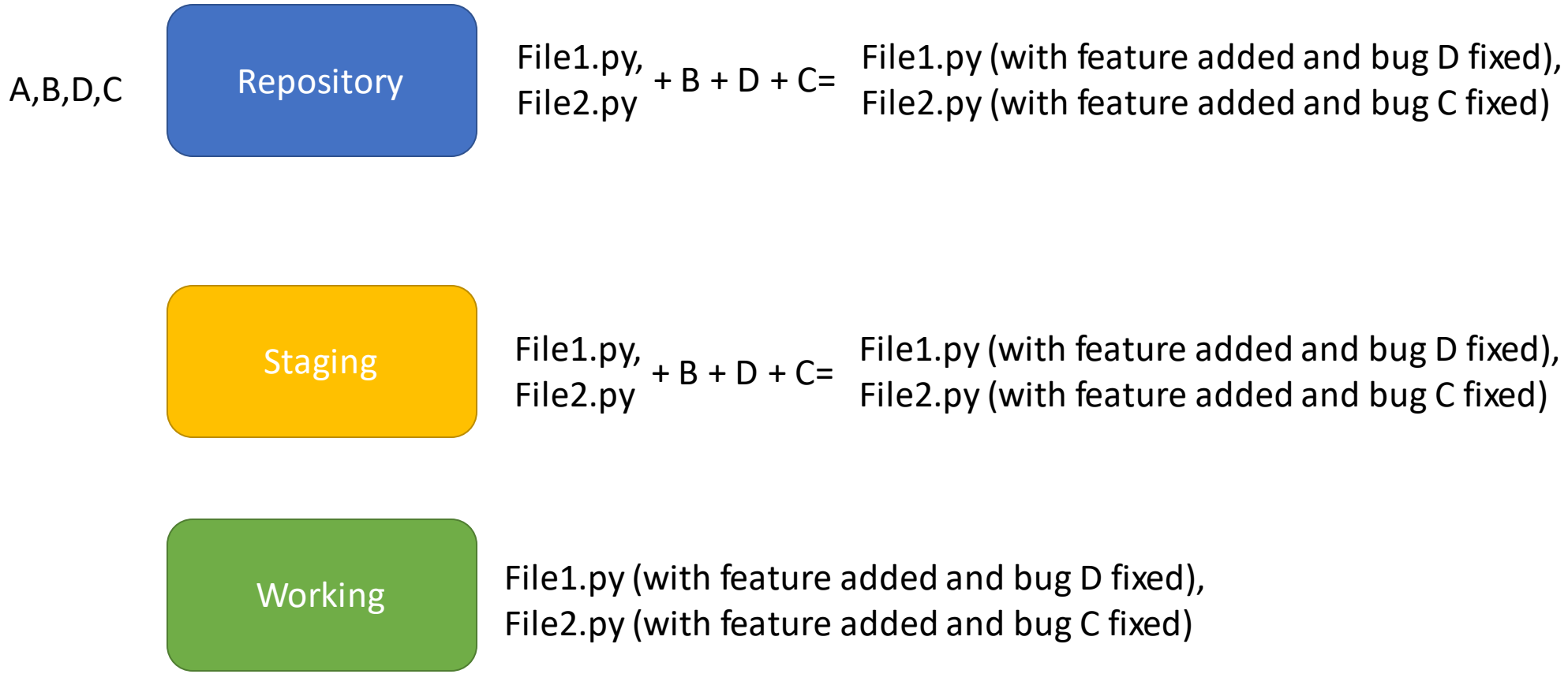
3 Tree Architecture, what's the point of add?



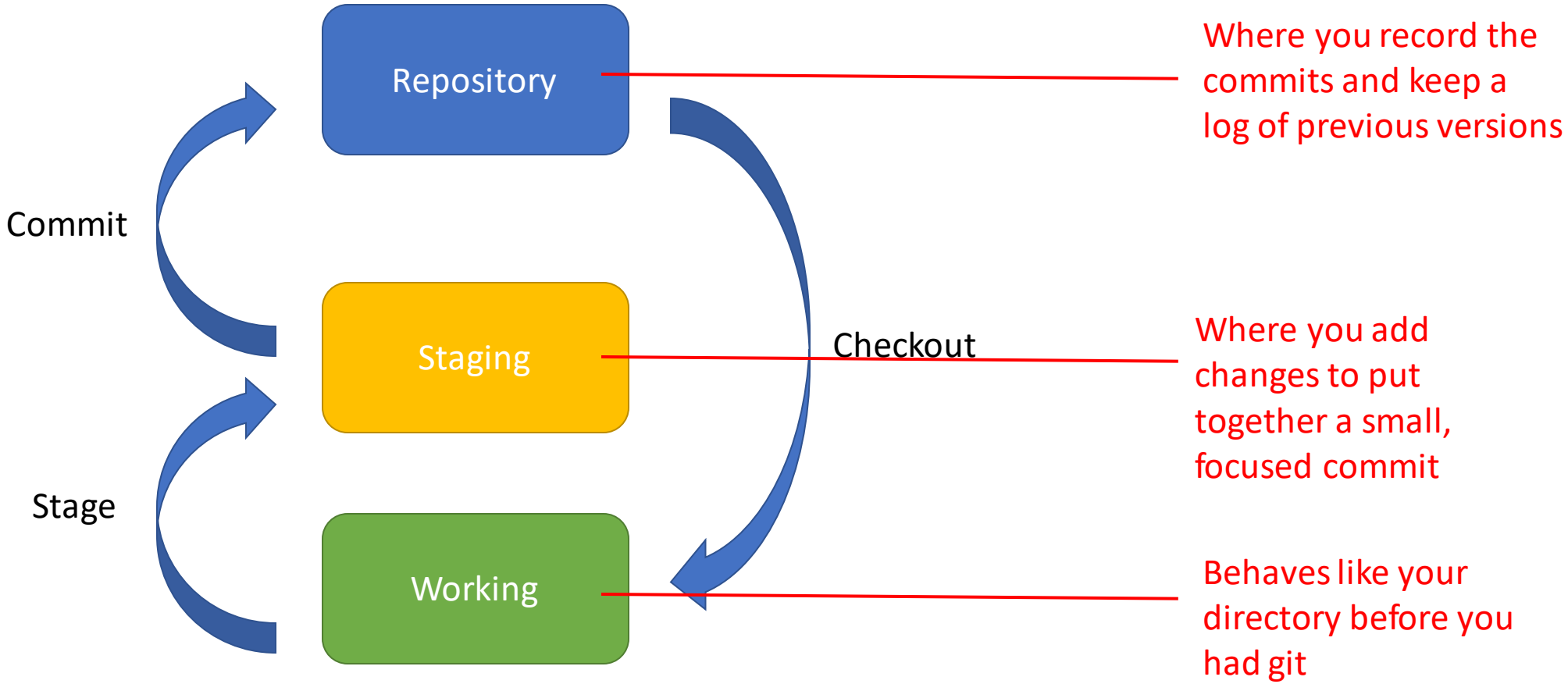
3 Tree Architecture, what's the point of add?



3 Tree Architecture, what's the point of add?



3 Tree Architecture



Commit messages

- Commit message should describe changes you are making in that change set
- You'll need to come back and look at change sets later
  - Want to just look at the commit message and know what's inside
- Short single-line summary (less than 50 characters)
- Optionally followed by a blank line and a more complete description

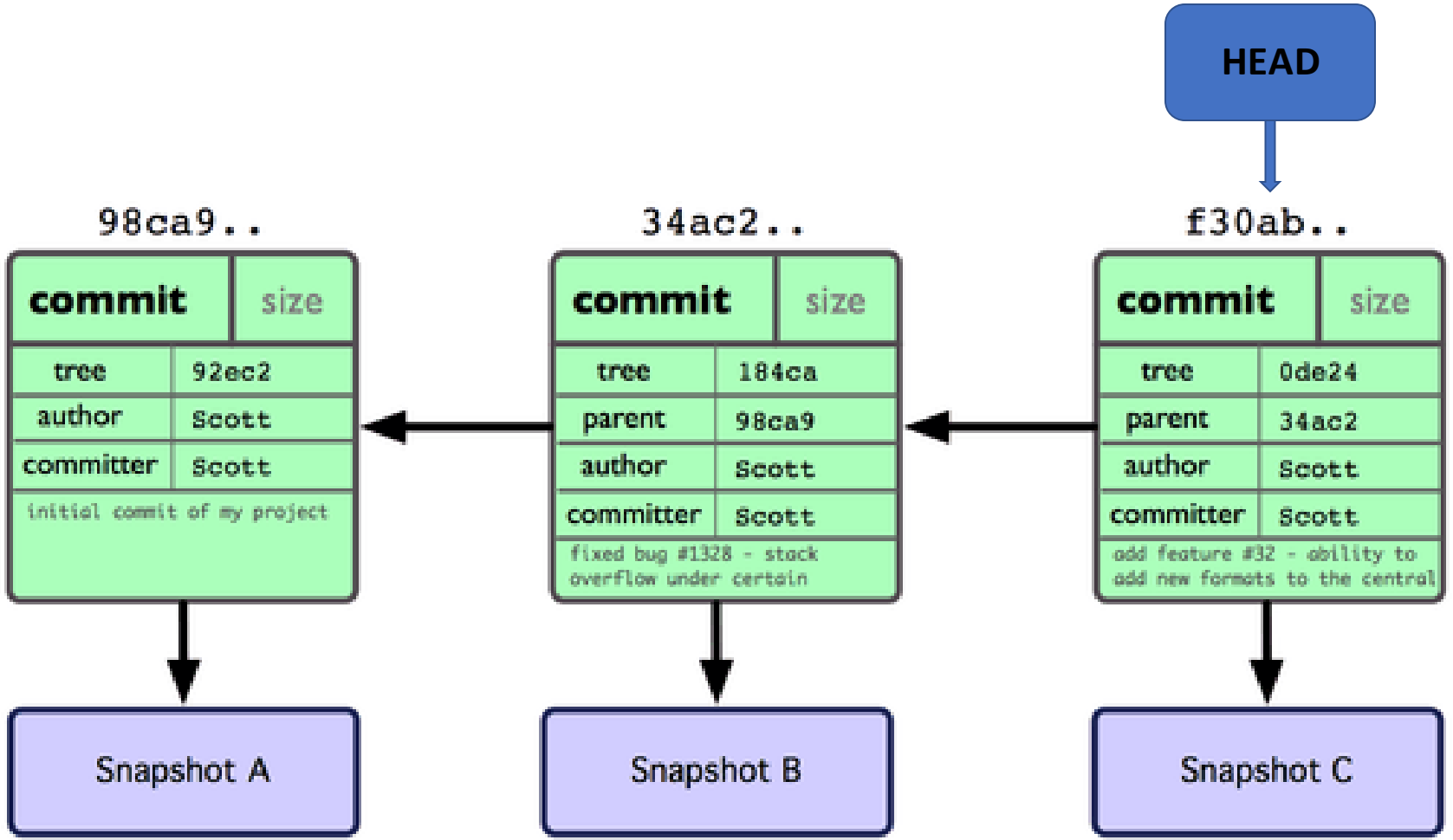
- Bad: "Fix typo"
  Good: "Adds missing semicolon to line 3"
- Bad: "updates nipype version, we should discuss if we want to keep using this version next week"

Some tools

- git status
  - Report back the difference between the work directory, staging index, and the repository
  - Untracked files
    - I can see your working directory has new things that I'm not currently tracking.
    - If you make text edits to these files, I won't be able to tell you what changed because I'm not tracking these.
- git log
- git diff
  - Compare working directory to repository

Undoing Changes

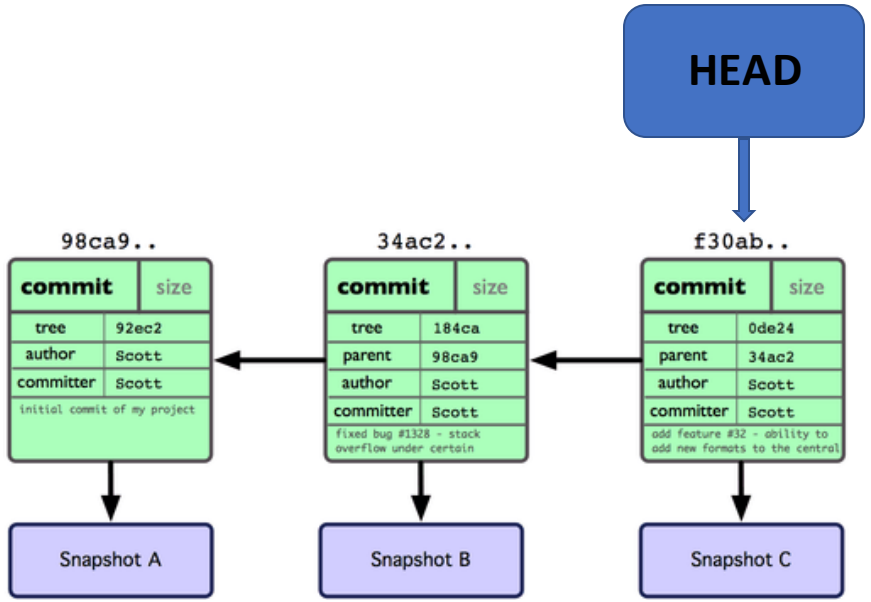
Repository record



Commands for undoing changes

- git checkout
  - Checkout file from repository and overwrite working directory

Checkout



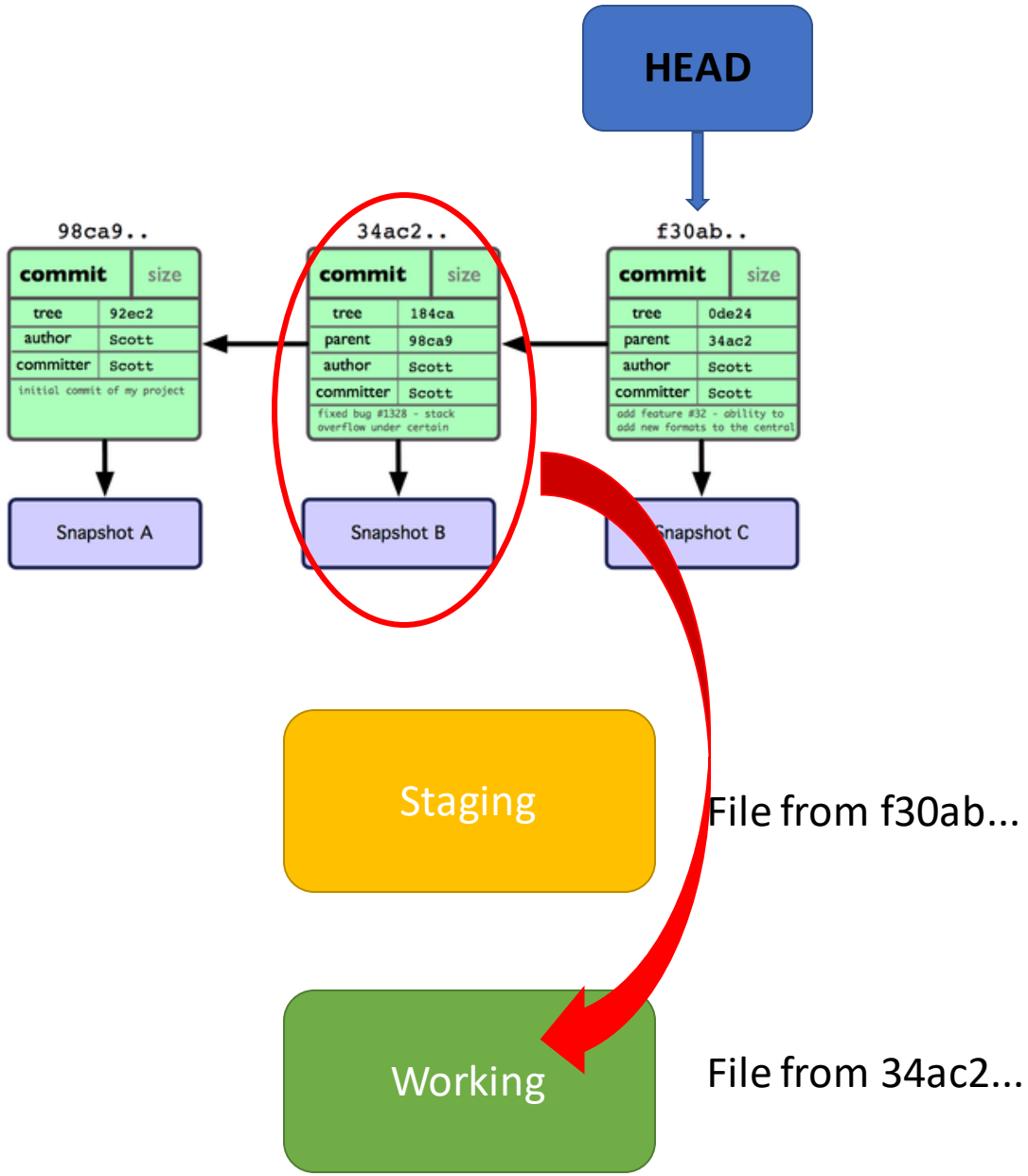
Staging

File from f30ab...

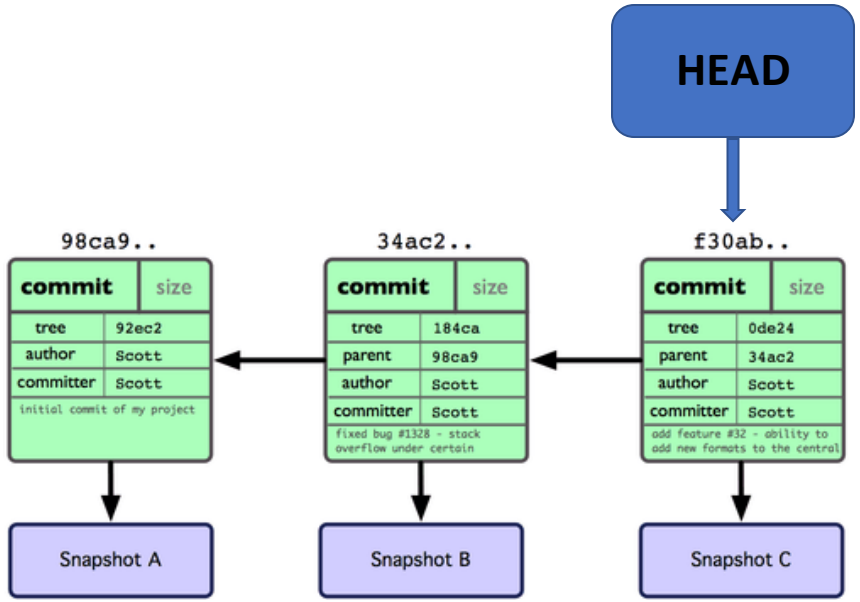
Working

File from f30ab...

Checkout


HEAD
98ca9..
commit
size
tree
92ec2
author
Scott
committer
Scott
initial commit of my project
34ac2..
commit
size
tree
184ca
parent
98ca9
author
Scott
committer
Scott
fixed bug #1328 - stack overflow under certain
f30ab..
commit
size
tree
0de24
parent
34ac2
author
Scott
committer
Scott
add feature #32 - ability to add new formats to the central
Snapshot A
Snapshot B
Snapshot C
Staging
File from f30ab...
Working
File from 34ac2...

Checkout



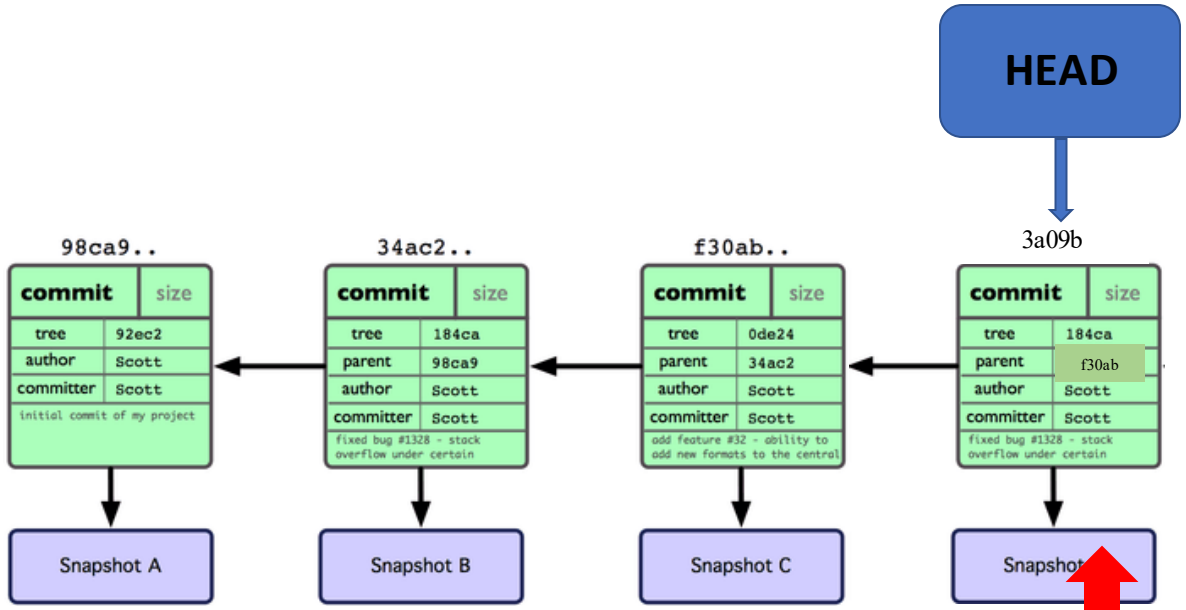
Staging

File from f30ab... + change set = File from 34ac2...

Working

File from 34ac2...

Checkout



Staging

File from f30ab... + change set = File from 34ac2...

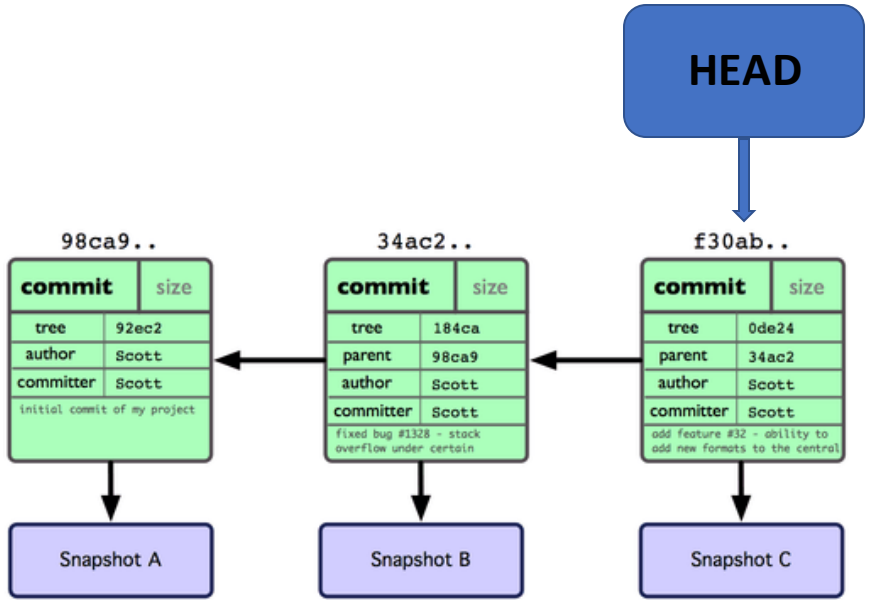
Working

File from 34ac2...

Commands for undoing changes

- git checkout
  - Checkout file from repository and overwrite working directory
- git reset
  - Move the repository head pointer and start recording from new position

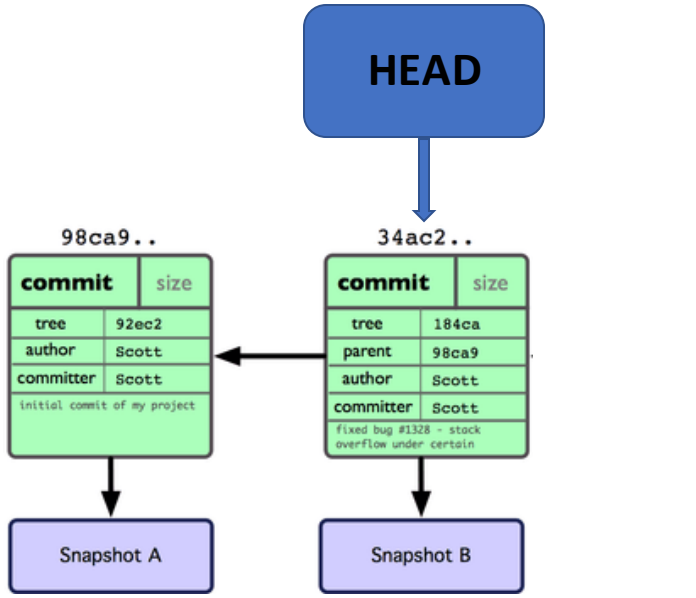
Reset


HEAD
98ca9..
commit
size
tree
92ec2
author
Scott
committer
Scott
initial commit of my project
34ac2..
commit
size
tree
184ca
parent
98ca9
author
Scott
committer
Scott
fixed bug #1328 - stack overflow under certain
f30ab..
commit
size
tree
0de24
parent
34ac2
author
Scott
committer
Scott
add feature #32 - ability to add new formats to the central
Snapshot A
Snapshot B
Snapshot C


Staging


Working

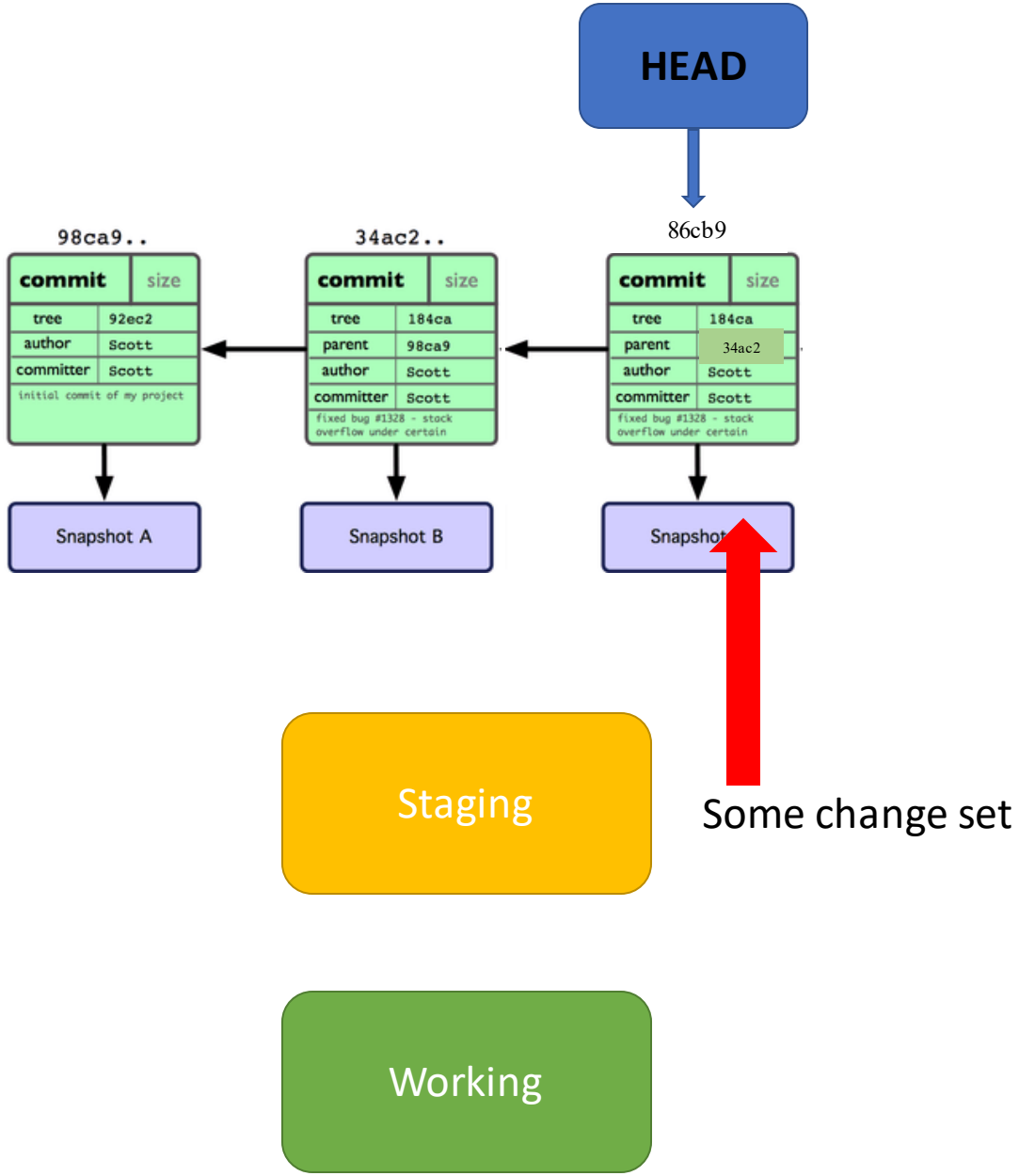
Reset



Staging

Working

Reset

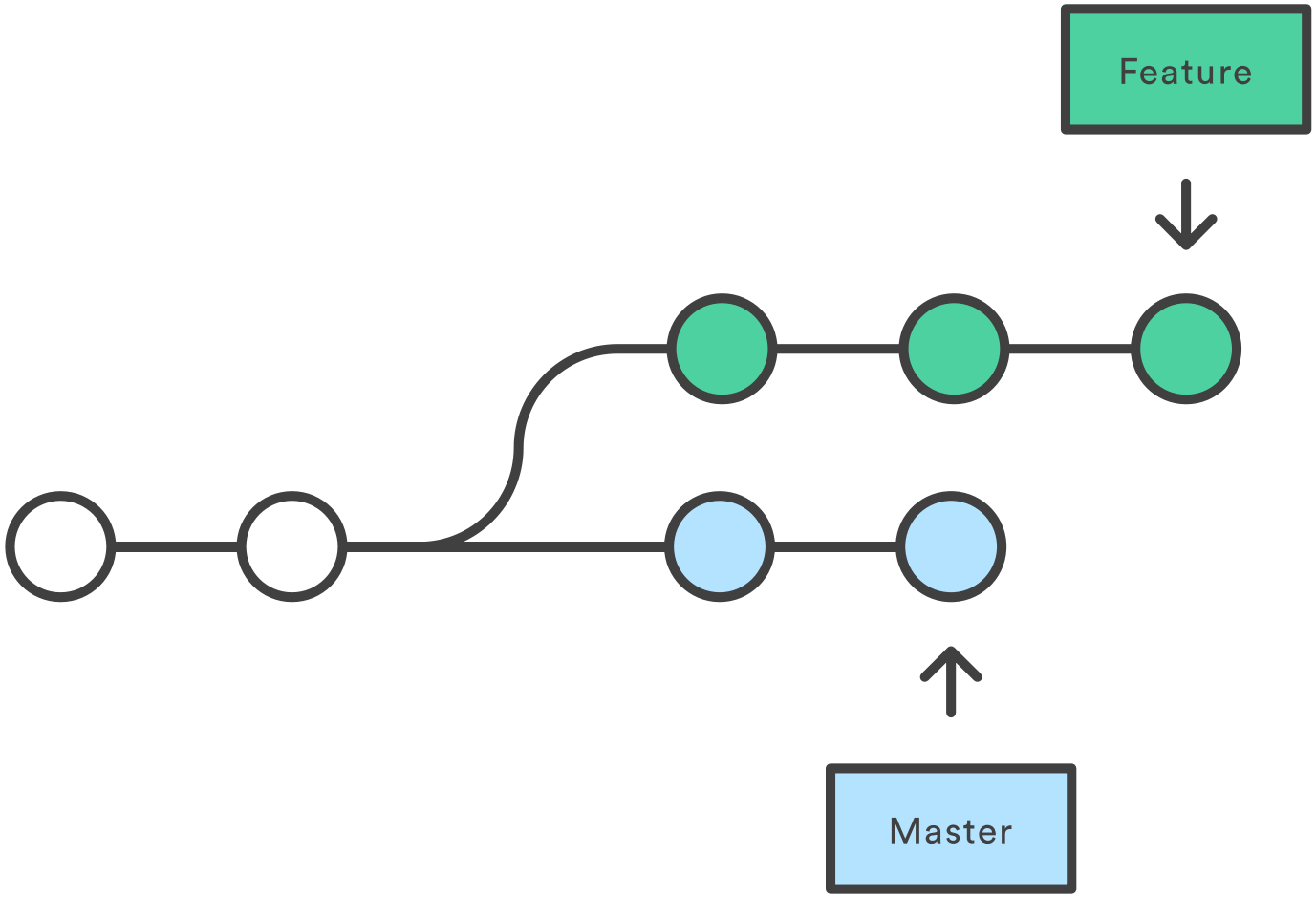

HEAD
98ca9..
commit
size
tree
92ec2
author
Scott
committer
Scott
initial commit of my project
34ac2..
commit
size
tree
184ca
parent
98ca9
author
Scott
committer
Scott
fixed bug #1328 - stack overflow under certain
86cb9
commit
size
tree
184ca
parent
34ac2
author
Scott
committer
Scott
fixed bug #1328 - stack overflow under certain
Snapshot A
Snapshot B
Staging
Some change set
Working

Commands for undoing changes

- git checkout
  - Checkout file from repository and overwrite working directory
- git reset
  - Move the repository head pointer and start recording from new position
- checkout vs reset
  - Committing after a checkout leaves the entire journey documented (documents that you tried something new and then reverted back to an older version)
  - Committing after a reset records over what you tried

Branch

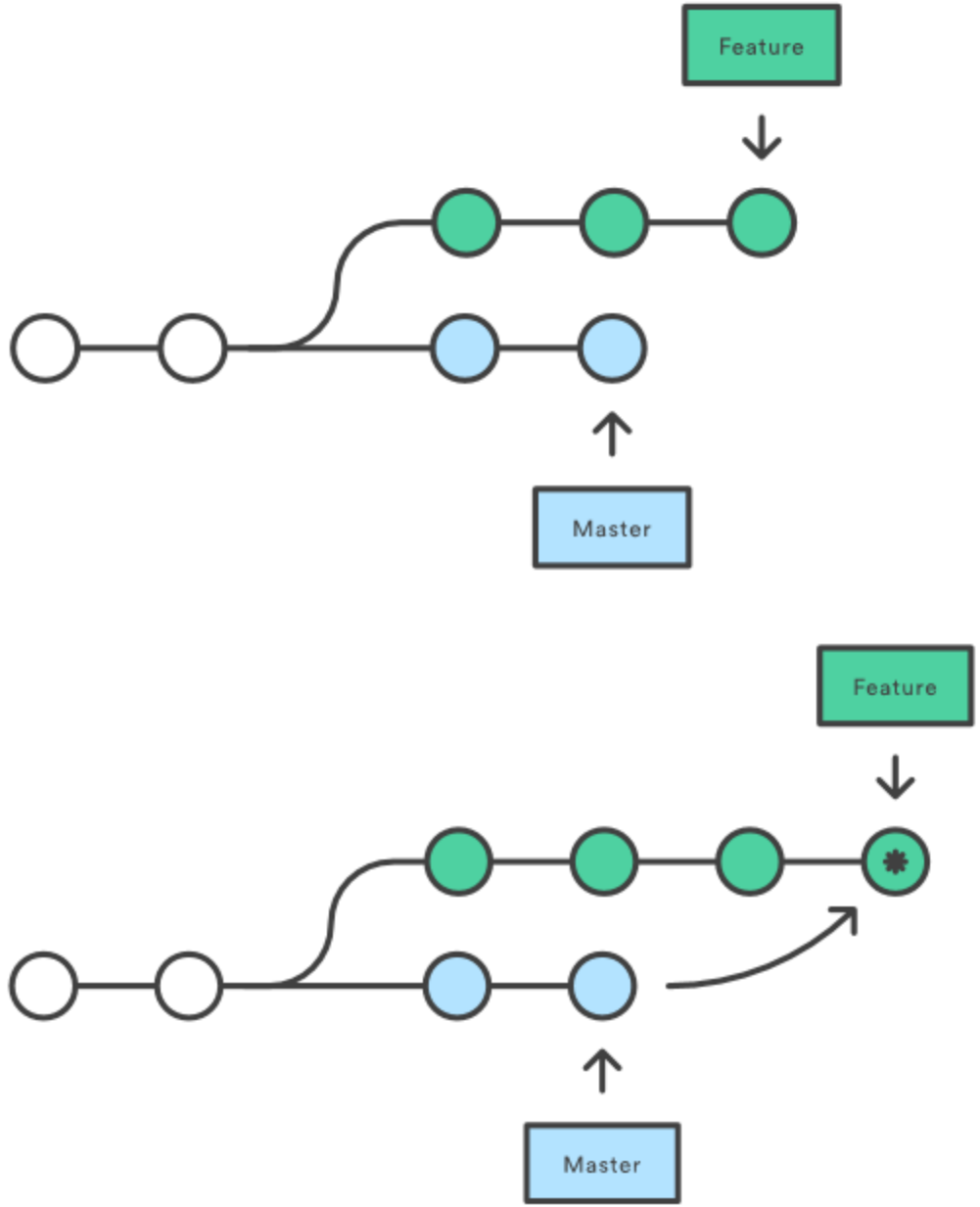
A forked commit history



When to branch off from the master branch?

- Branching from master
- Fix a bug
- Add a feature
- Try something new, development
- Refactor code

Merge

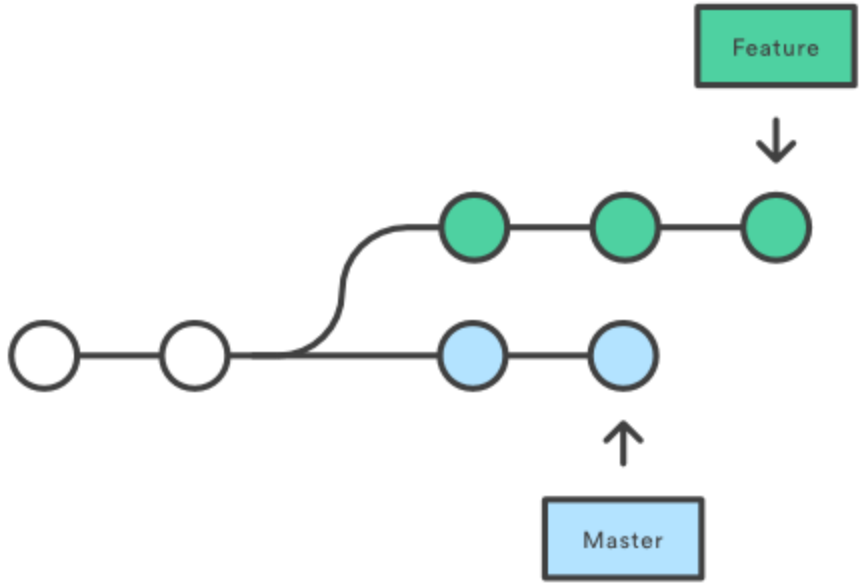

Feature
Master
Feature
Master
Merge Commit

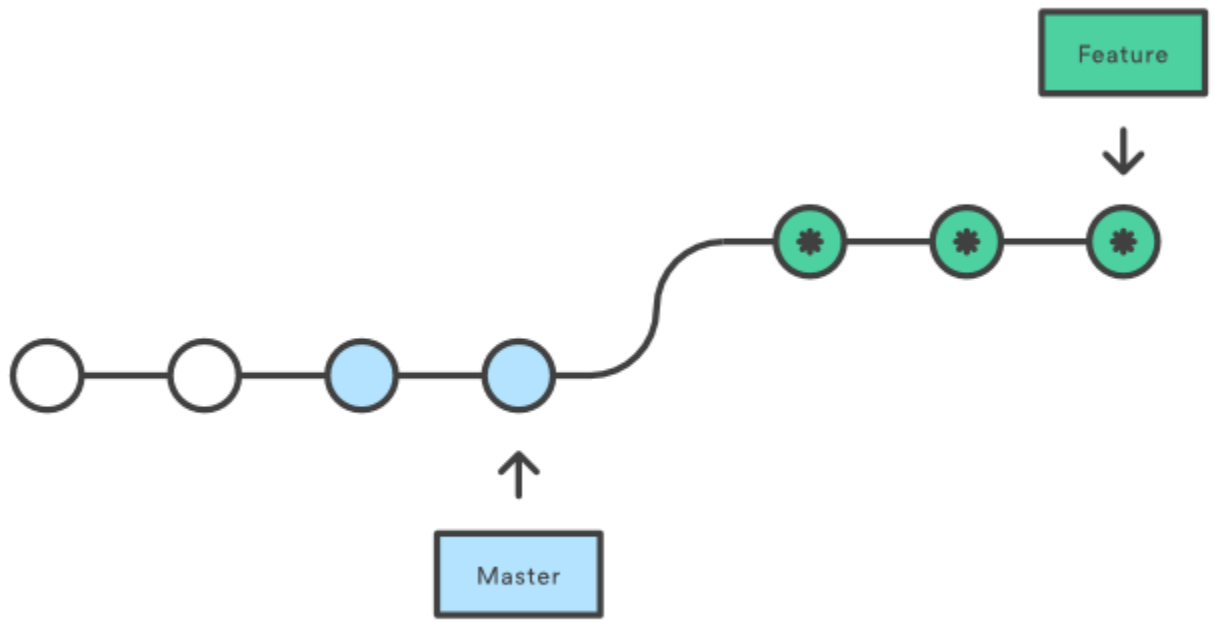
When to merge

- Update master branch with new functionality from a feature branch
- Feature branch track incremental changes in master branch

Conflicts!

Rebase



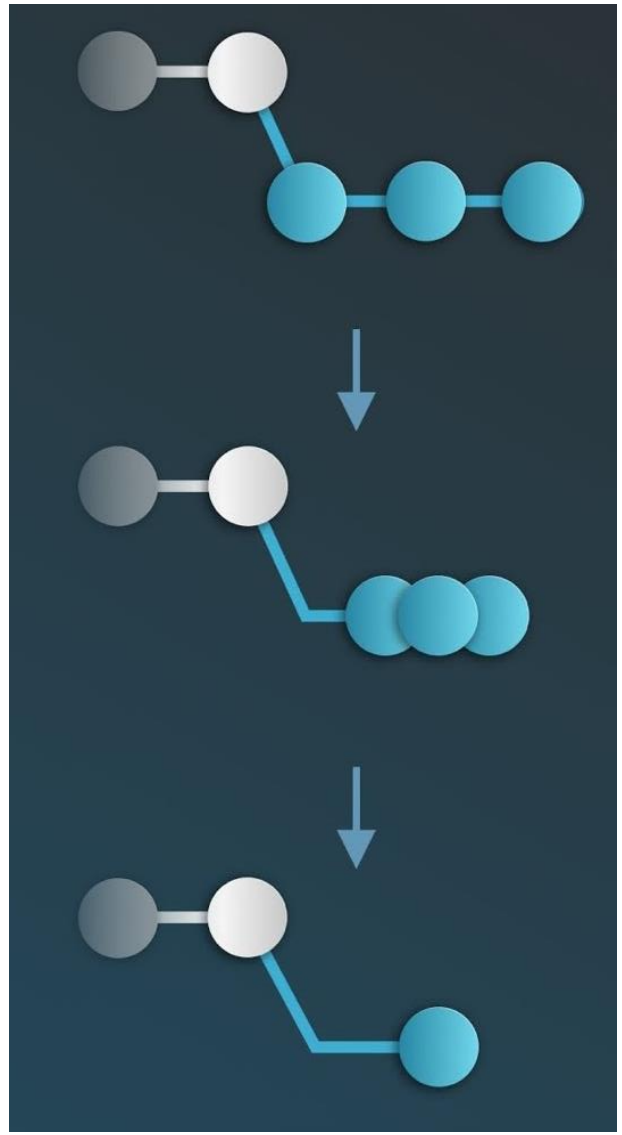


Brand New Commit

When to rebase

- Feature branch track incremental changes in master branch
- Want a clean logs and a linear history (different than merge)
- It's destructive, sha changes, commits are rewritten, tricky to undo

Squash



When to squash

- Fold two or more commits into one
- Prepare feature branch to be merged into master:
- Consolidate log
- Package up the commits so that code is in working condition.

Branching workflow

- Branch to try something new!
- Use rebase to incorporate any changes made in the master branch into your new feature branch
- When the feature branch is complete, clean up the history using squash (squash intermediate commits that are not in a working state, combine groups of commits)
- Merge the new feature into the master branch

Remotes and clone

Remotes and clone

- Remote: versions of your project hosted on the internet
  - Github
  - Gitlab
  - CFMM
- Clone: The command used to retrieve the repository

More resources

- Videos with tutorials and worked out examples
  - Linkedin learning: https://linkedinlearning.uwo.ca/
    Git essentials with Kevin Skoglund

- Git GUIs
  - Sourcetree
  - Smartgit

- Practice on fake files first!

- Come to Igor's tutorial on Friday

Interactive Staging

- Allows you to stage portions of files, interactively
- Even smaller, more focused commits
- Like a little program that lets you stage only the things you want
- Sometimes you fix multiple bugs at once, but you don't want those fixes to be grouped together in your log