Universität Stuttgart

J. Valentin
F. Lindner

IPVS

SGS

SimTech
Cluster of Excellence

The Gist of Git – A Git Overview Pointlessly Littered with Memes

Numerische Simulation

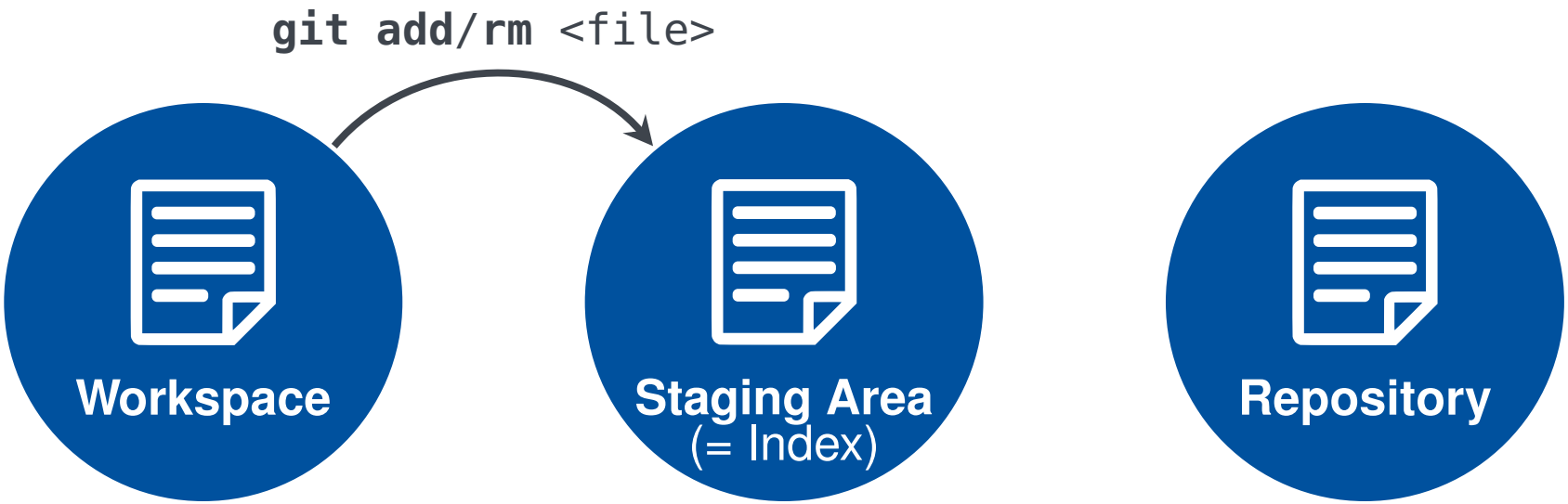
Nov 2, 2016


ONE DOES NOT SIMPLY
LEARN GIT
imgflip.com

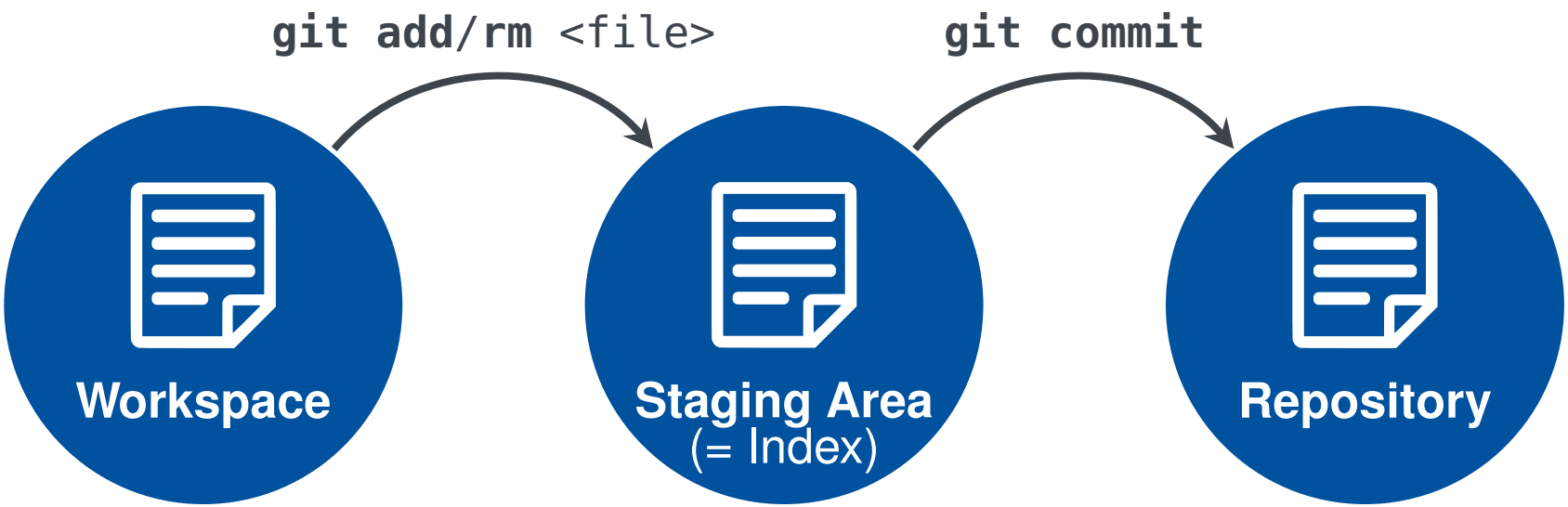
About Commits

Workspace, Staging Area, Repository

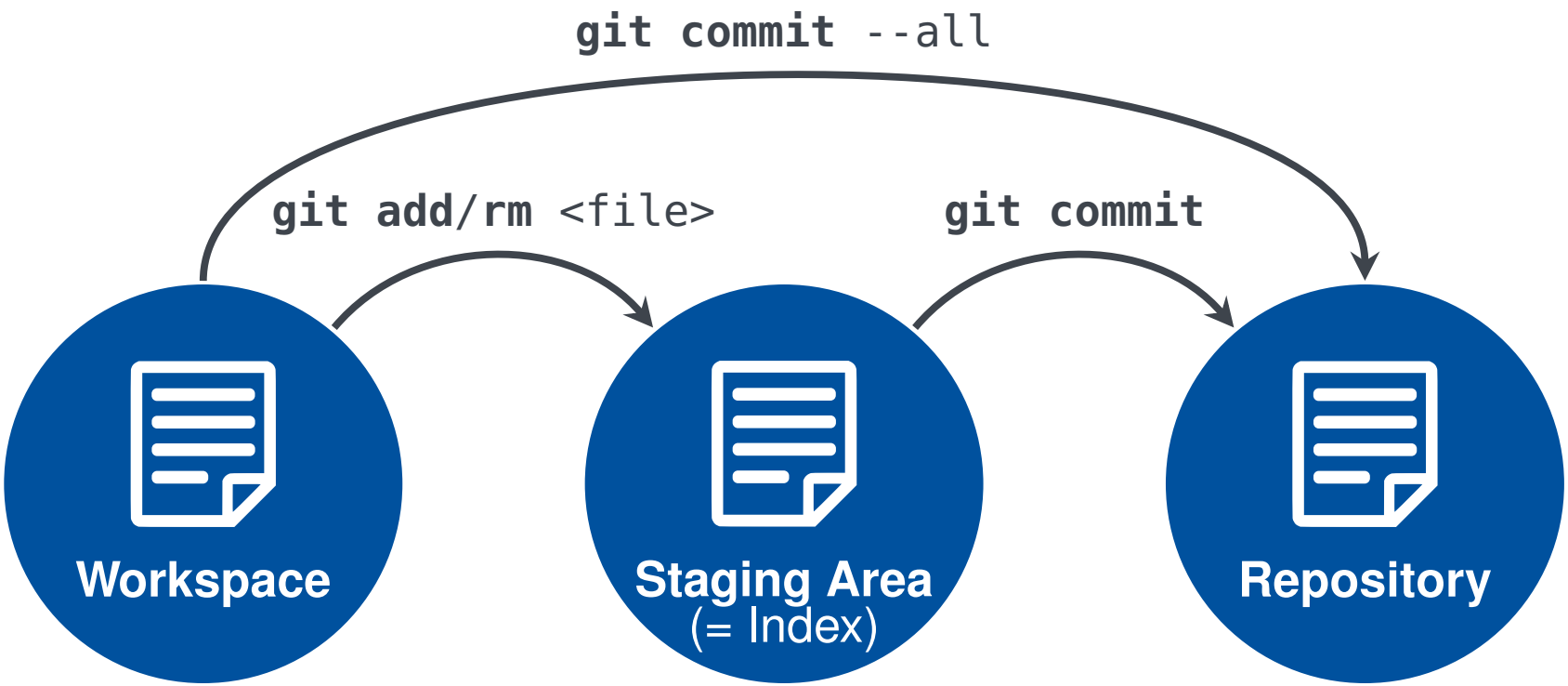
Workspace, Staging Area, Repository



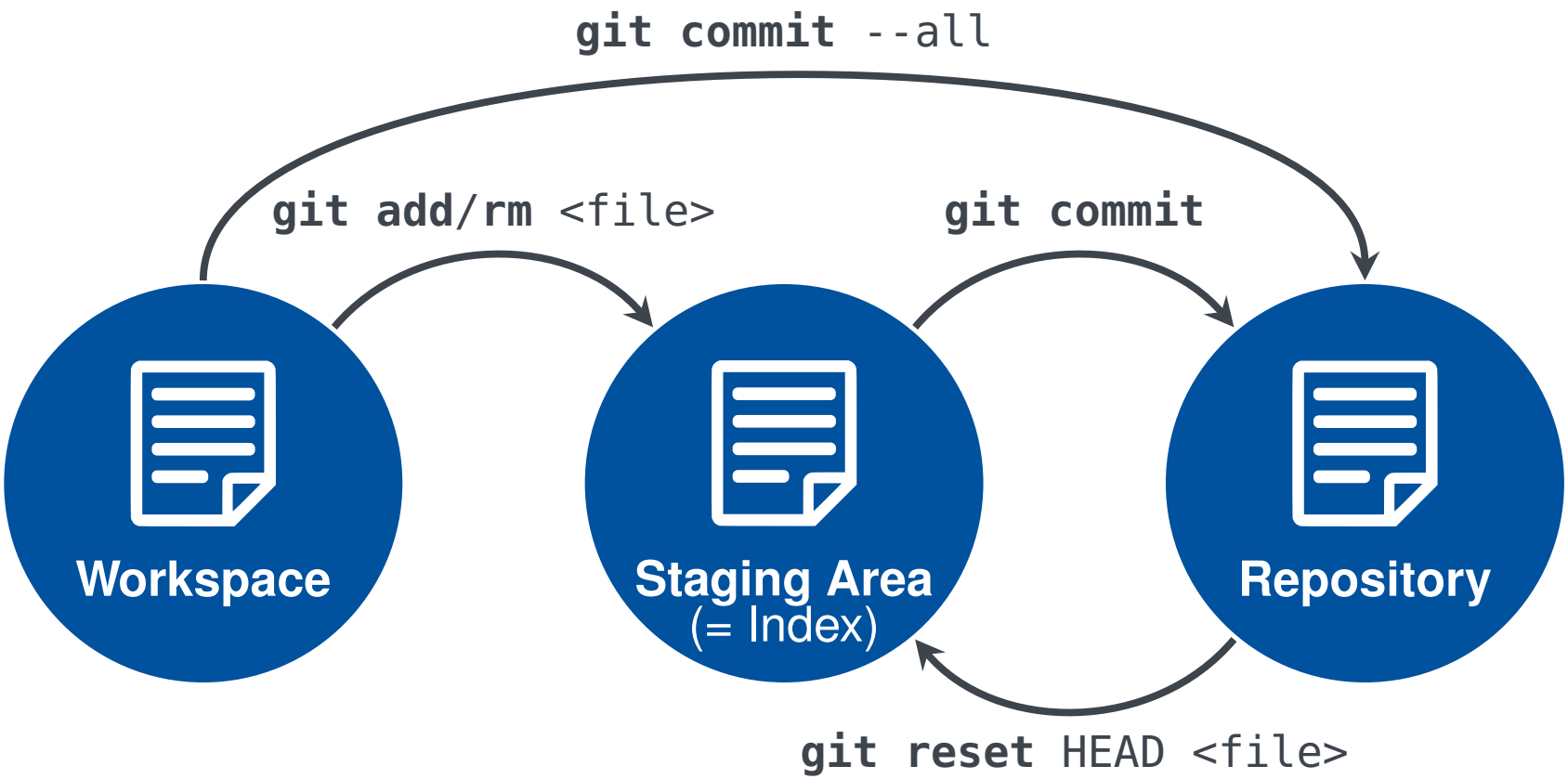
Workspace, Staging Area, Repository



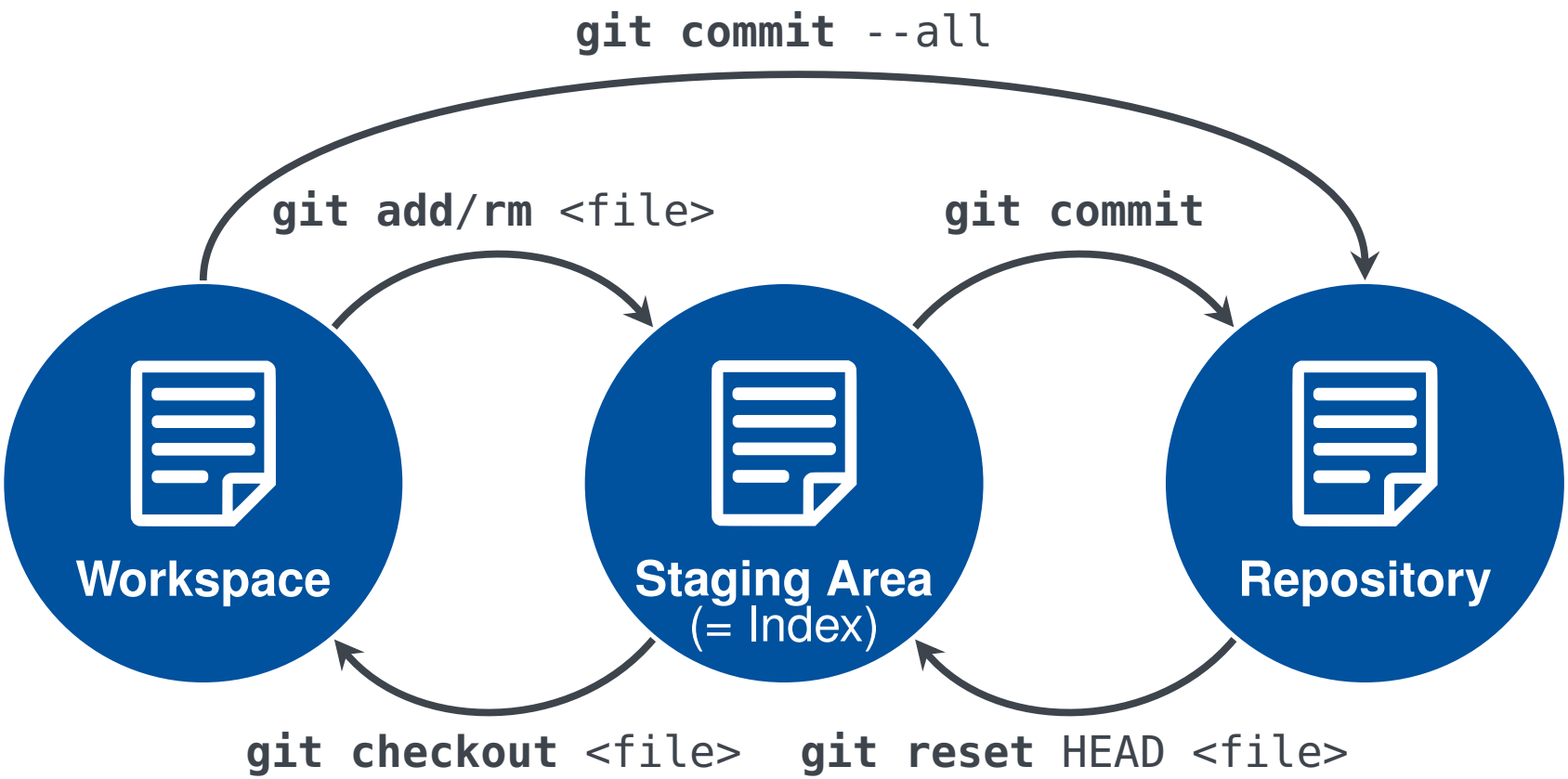
Workspace, Staging Area, Repository



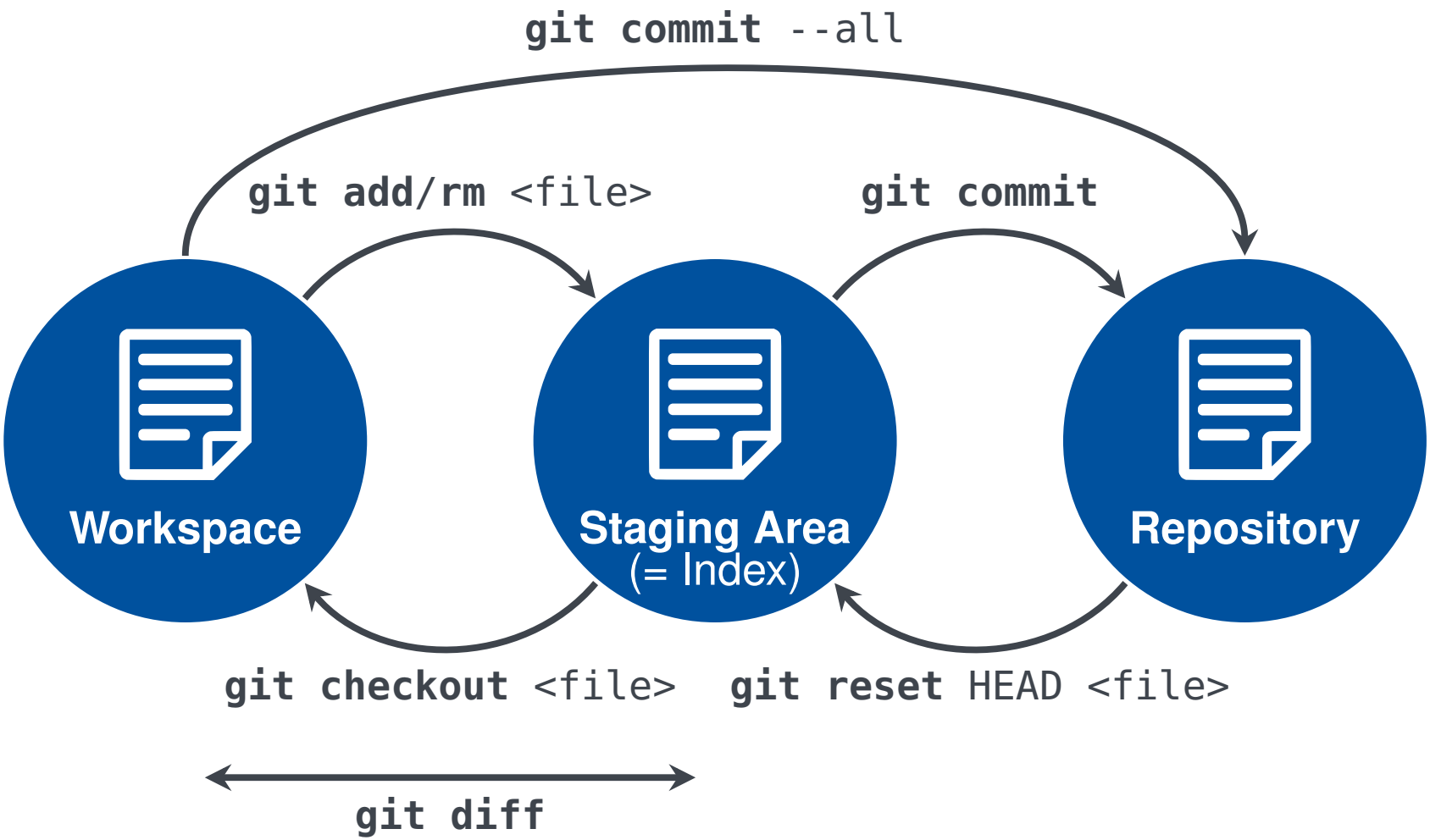
Workspace, Staging Area, Repository



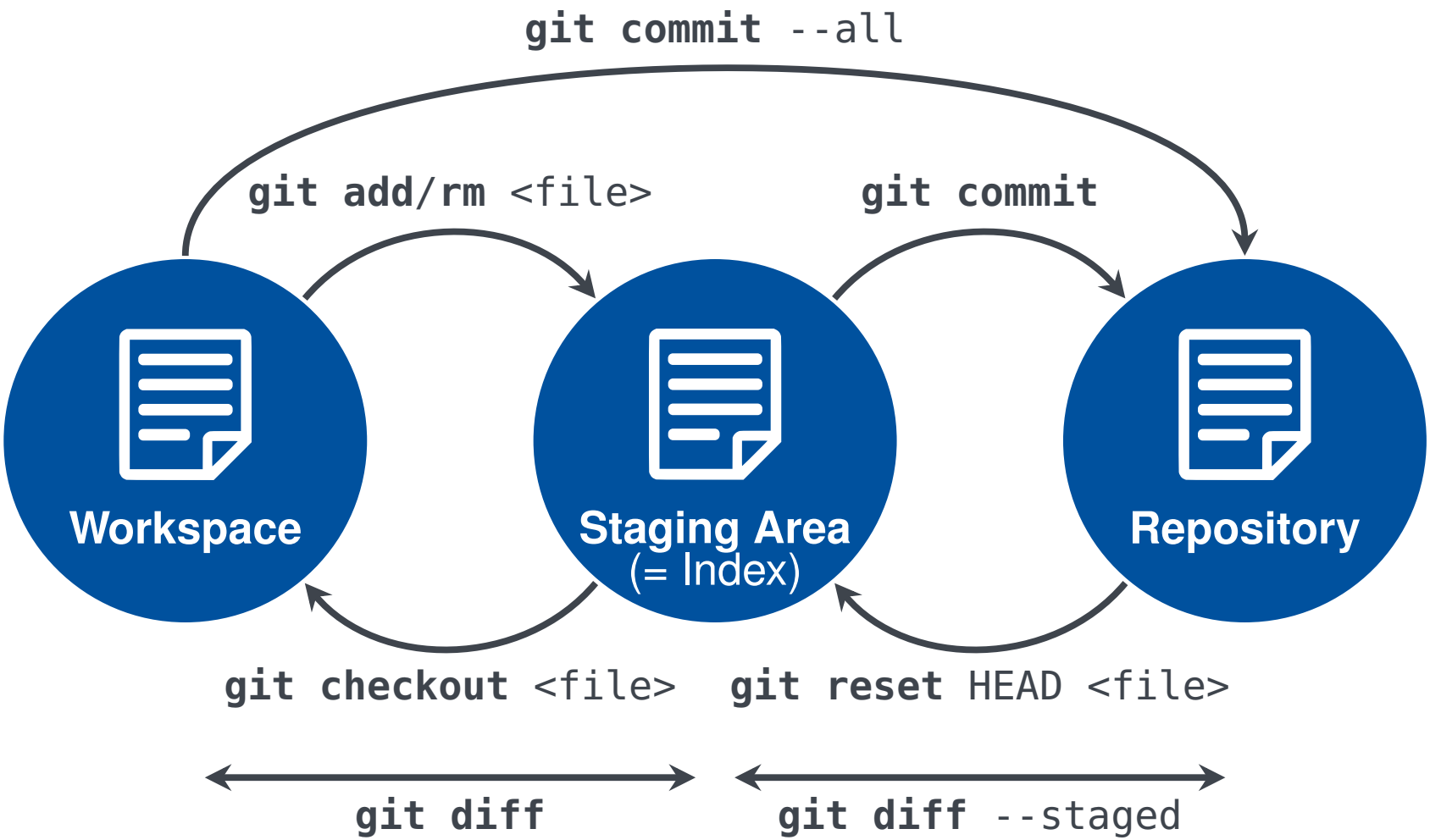
Workspace, Staging Area, Repository



Workspace, Staging Area, Repository



Workspace, Staging Area, Repository



Status of Workspace and Staging Area

`git status`: Show status of workspace and staging area

- *Changes to be committed:*
  Change is in workspace and staging area
- *Changes not staged for commit:*
  Change is in workspace, but not in staging area
- *Untracked files:*
  New file is in workspace, but not in staging area

Interactive Add

- **`git add --interactive`:** Change staging area interactively
  - `update`: Add changes of a file to staging area
  - `revert`: Remove changes of a file from staging area
  - `patch`: Add individual hunks to staging area (choose which changed lines of a file to add)

```
$ git add --interactive
           staged     unstaged path
1:          +0/-1      nothing TODO
2:          +1/-1      nothing index.html
3:    unchanged         +5/-1 lib/simplegit.rb

*** Commands ***
1: status     2: update      3: revert     4: add untracked
5: patch      6: diff        7: quit       8: help
What now>
```

Interactive Add

- **`git add --interactive`:** Change staging area interactively
  - `update`: Add changes of a file to staging area
  - `revert`: Remove changes of a file from staging area
  - `patch`: Add individual hunks to staging area (choose which changed lines of a file to add)

```
$ git add --interactive
           staged     unstaged path
1:          +0/-1      nothing TODO
2:          +1/-1      nothing index.html
3:    unchanged        +5/-1 lib/simplegit.rb

*** Commands ***
1: status     2: update      3: revert      4: add untracked
5: patch      6: diff        7: quit        8: help
What now>
```

Interactive Add

- **`git add --interactive`:** Change staging area interactively
  - `update`: Add changes of a file to staging area
  - `revert`: Remove changes of a file from staging area
  - `patch`: Add individual hunks to staging area (choose which changed lines of a file to add)

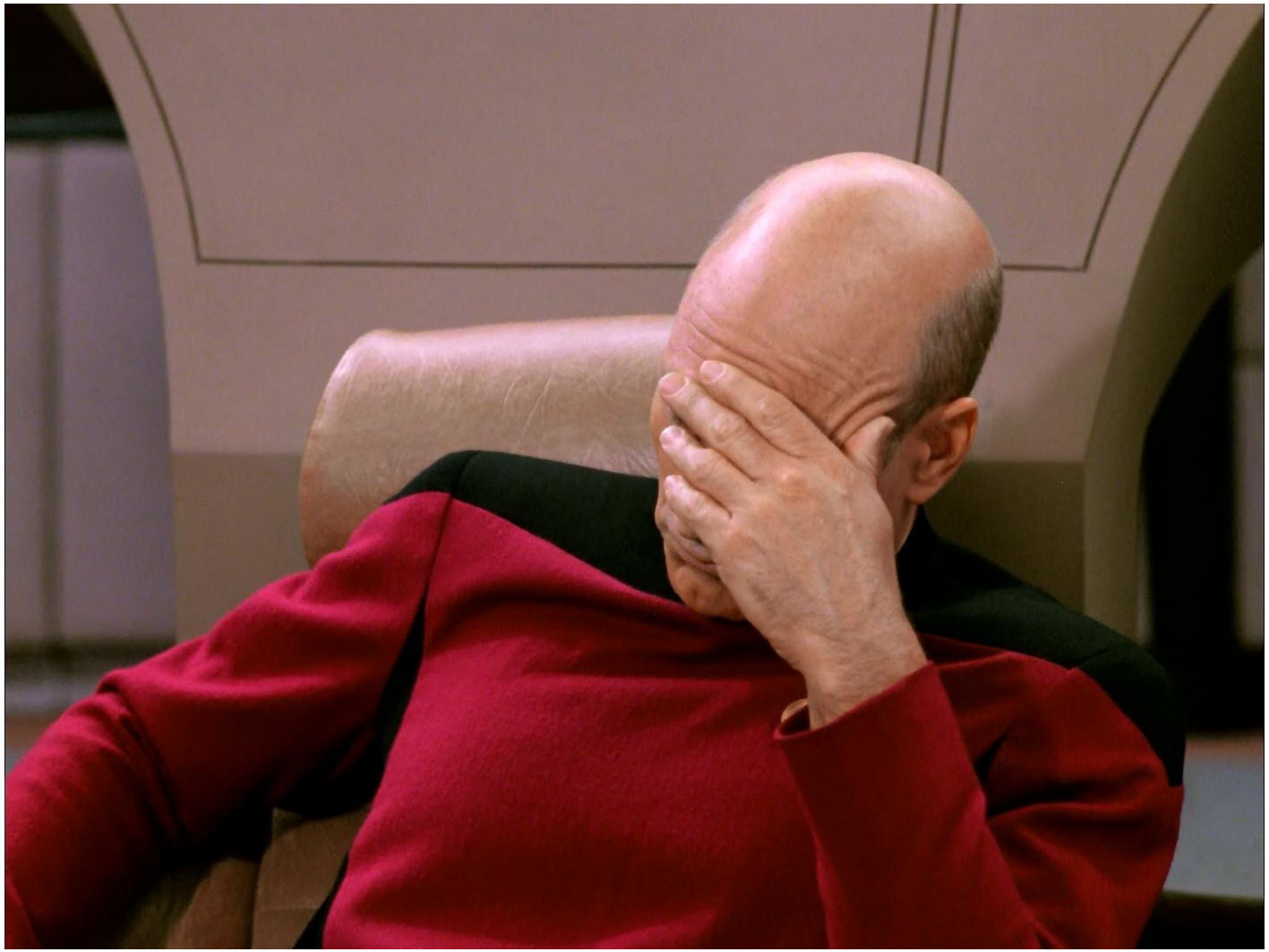
```
$ git add --interactive
           staged     unstaged path
1:          +0/-1      nothing TODO
2:          +1/-1      nothing index.html
3:    unchanged         +5/-1 lib/simplegit.rb

*** Commands ***
1: status     2: update      3: revert     4: add untracked
5: patch      6: diff        7: quit       8: help
What now>
```

How to Write Good Commit Messages

xkcd by Randall Munroe, licensed under CC BY-NC 2.5 Generic

How to Write Good Commit Messages



How to Write Good Commit Messages

Seven rules for a good commit message
(from `http://chris.beams.io/posts/git-commit/`):

```
Summarize changes in 50 characters or less

Separate subject from body with a blank line. Capitalize the
subject line. Don't end the subject line with a period. Use the
imperative mood in the subject line. Wrap the body at 72
characters. Use the body to explain what and why vs. how.
```

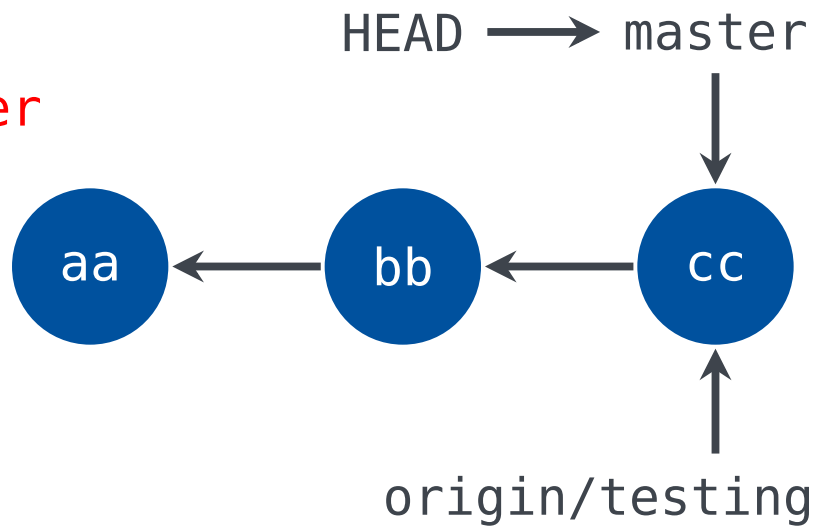
All Things Branches


WHAT IF I TOLD YOU
GIT BRANCHES ARE JUST NAMED COMMITS
imgflip.com

Branching

- Every commit knows its parent(s)
- **Branch:** pointer to a commit
- **`HEAD`:** pointer to a branch (or to a commit, “detached `HEAD`”), branch is automatically moved forward on committing, change branch with **`git checkout`** `<branch>`

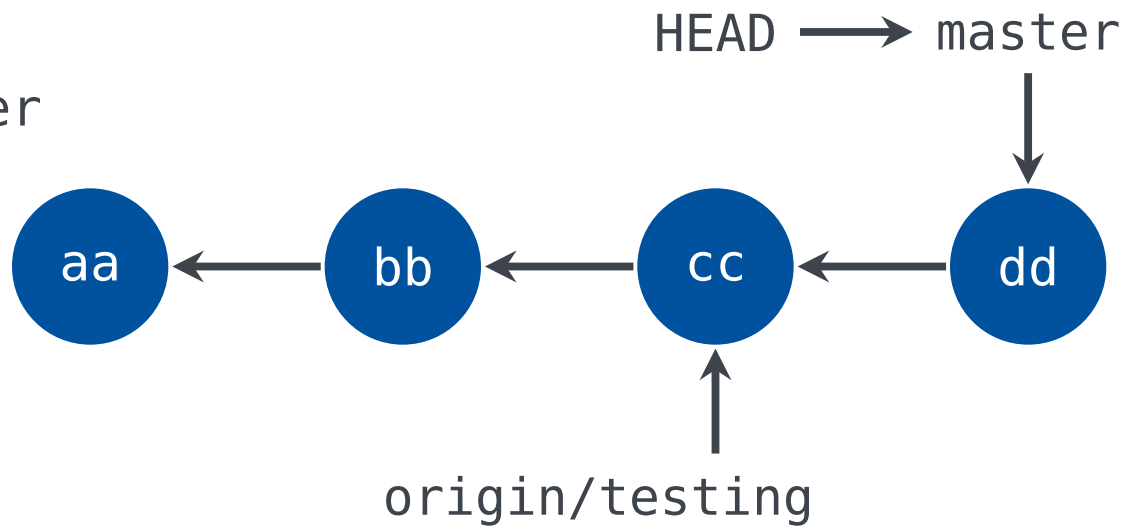
```
git checkout master
git commit
git fetch
```



Branching

- Every commit knows its parent(s)
- **Branch:** pointer to a commit
- **`HEAD`:** pointer to a branch (or to a commit, “detached `HEAD`”), branch is automatically moved forward on committing, change branch with **`git checkout`** `<branch>`

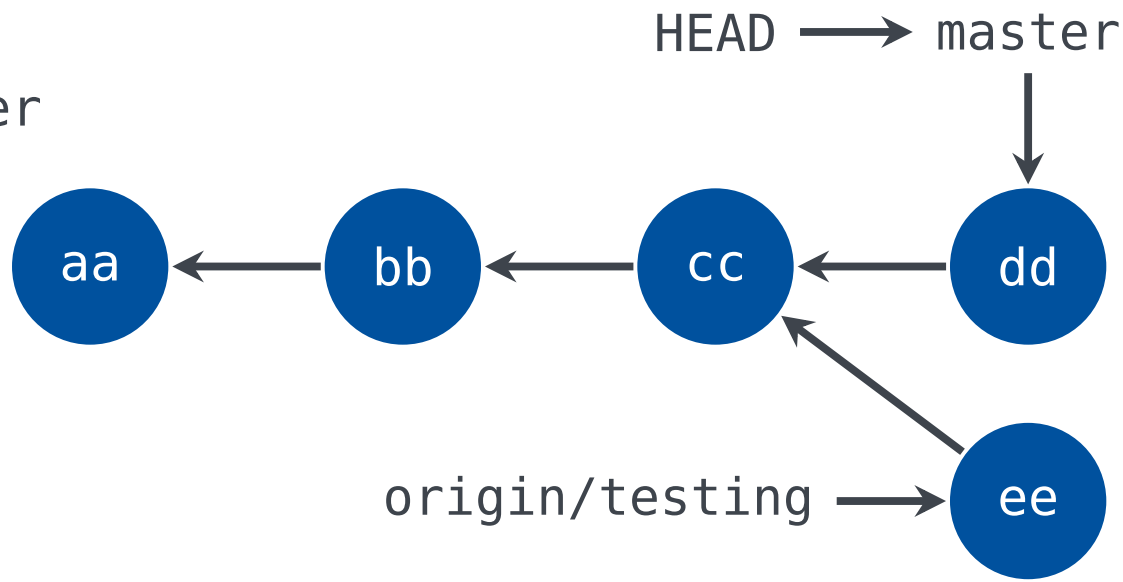
```
git checkout master
git commit
git fetch
```



Branching

- Every commit knows its parent(s)
- **Branch:** pointer to a commit
- **`HEAD`:** pointer to a branch (or to a commit, “detached `HEAD`”), branch is automatically moved forward on committing, change branch with **`git checkout`** `<branch>`

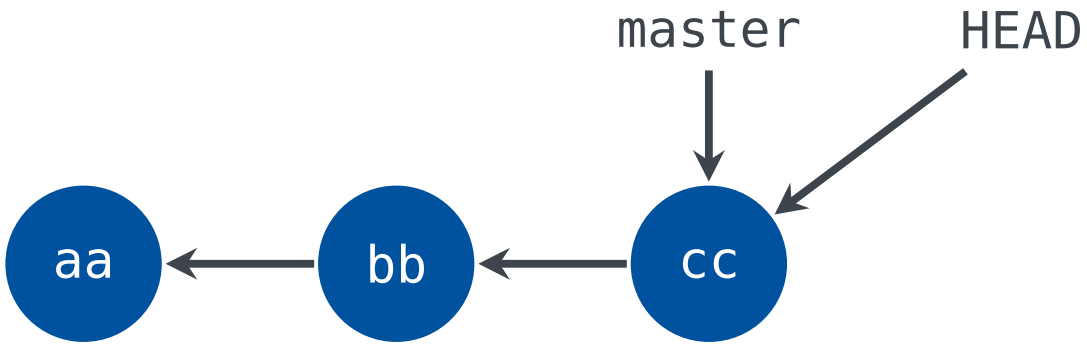
```
git checkout master
git commit
git fetch
```



Detached HEAD

- **Detached HEAD:** HEAD points to a commit instead of a branch

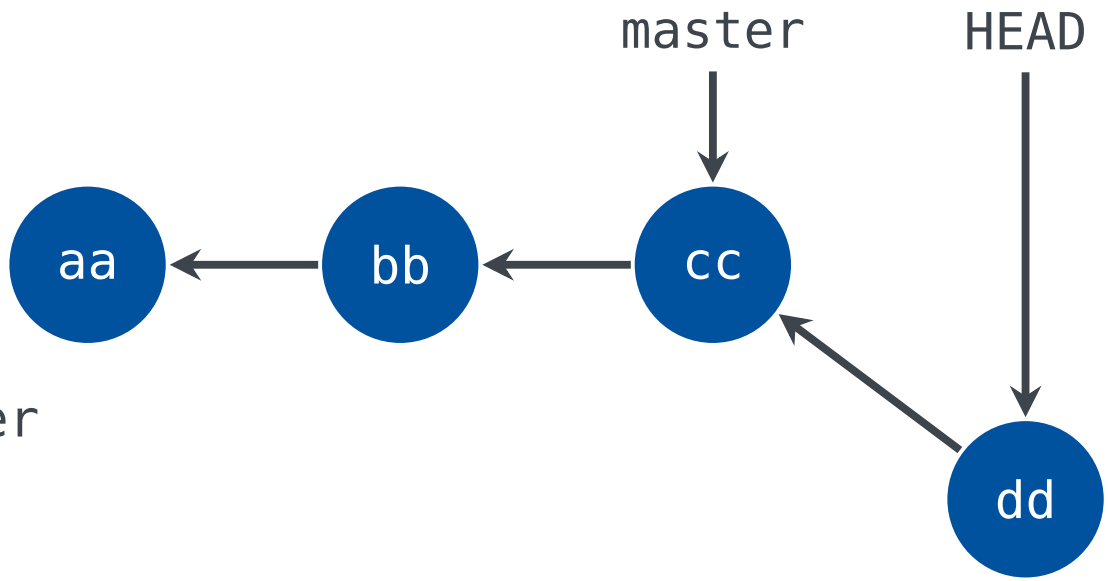
```
git checkout cc
git commit
git checkout master
git commit
```



Detached HEAD

- **Detached HEAD:** HEAD points to a commit instead of a branch

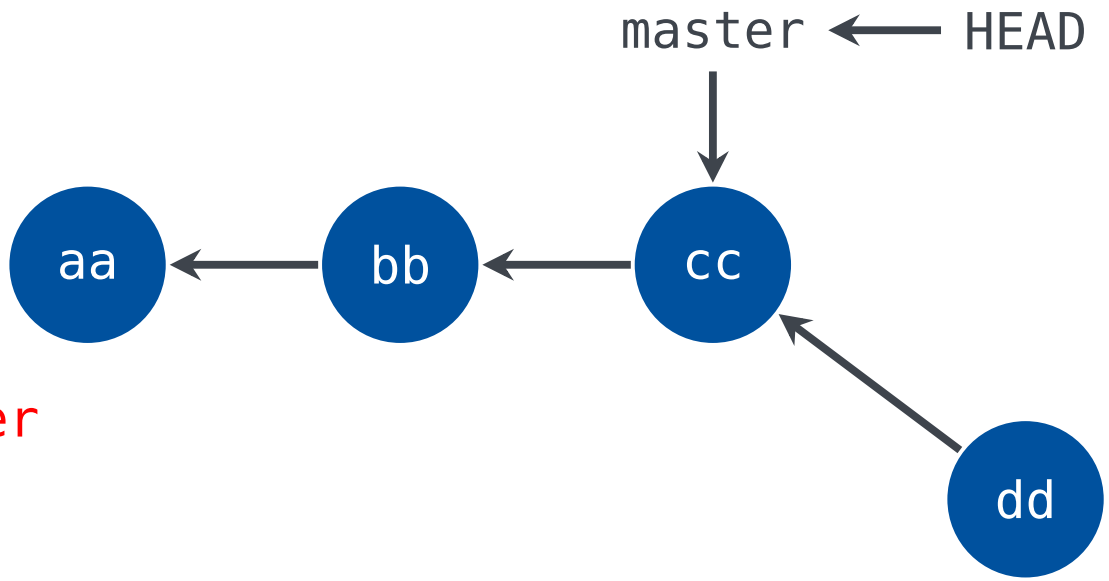
```
git checkout cc
git commit
git checkout master
git commit
```



Detached HEAD

- **Detached HEAD:** HEAD points to a commit instead of a branch

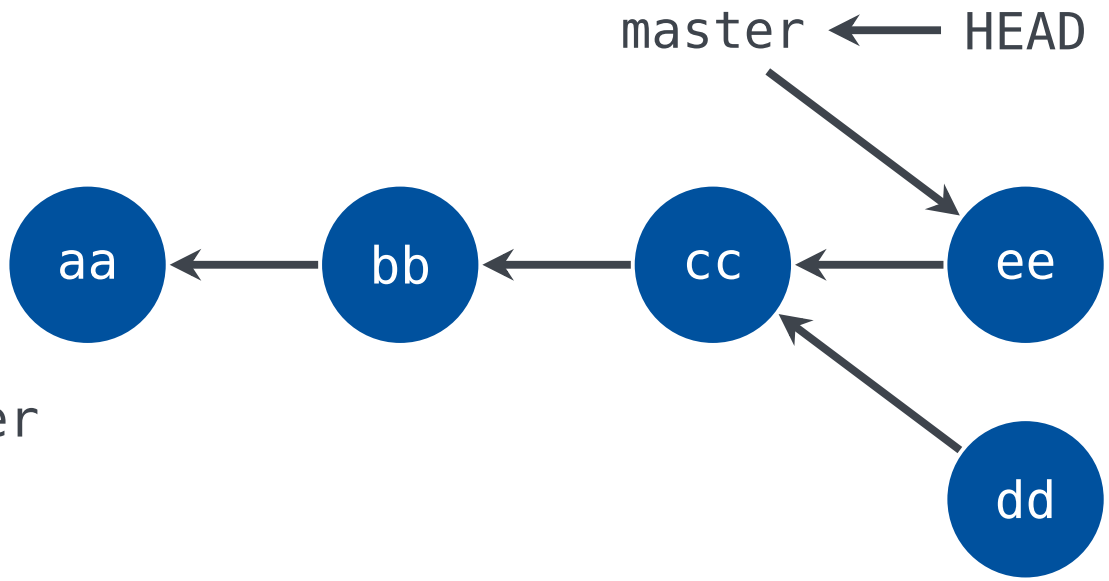
```
git checkout cc
git commit
git checkout master
git commit
```



Detached HEAD

- **Detached HEAD:** HEAD points to a commit instead of a branch

```
git checkout cc
git commit
git checkout master
git commit
```



Branching

- List branches with **`git branch --list -avv`**

```
  master                  fc7f1ed [origin/master] Merge sgopt in...
  peter_schober           89341d4 Add safety check in SubspaceRe...
* sgopt                   f8b42cf [origin/sgopt] Evaluate functi...
  remotes/origin/master   fc7f1ed Merge sgopt into master
  remotes/origin/sgopt    f8b42cf Evaluate function in Operation...
```

- Terminology:
  - **(Local) branch:** Branch in your repo
  - **Remote branch:** Branch in someone else's repo
  - **Remote-tracking branch:** Special local branch indicating remote branch on last **`fetch`**, can't be checked out
  - **Upstream branch:** link between a local and a remote-tracking branch, used for **`pull`**/**`push`**

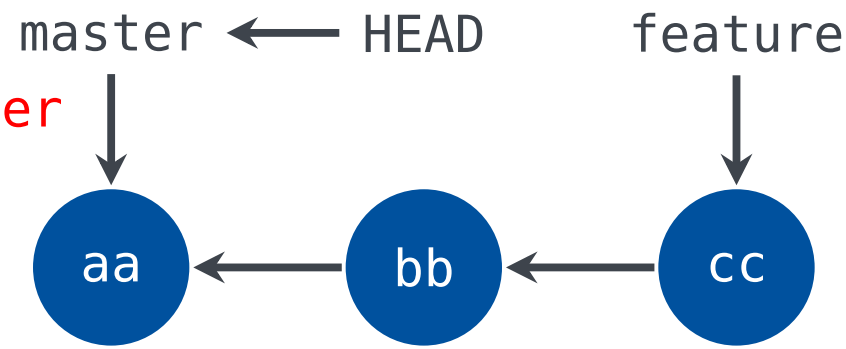
Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  1. One commit is ancestor of the other

Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  1. One commit is ancestor of the other

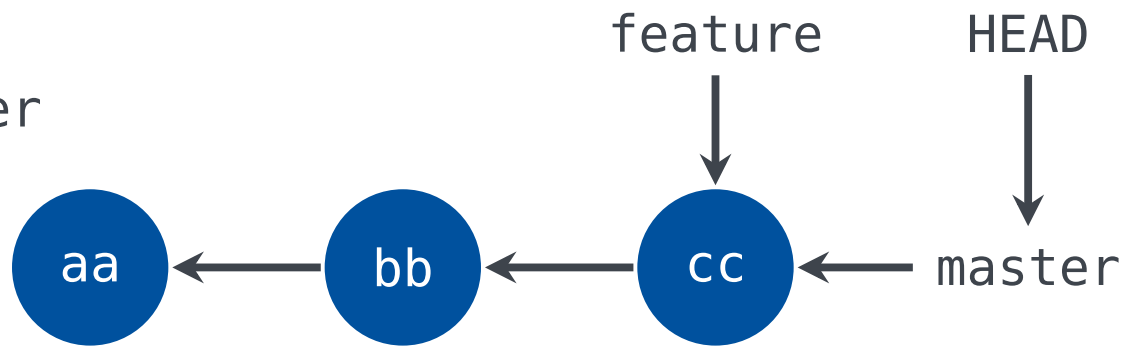
```
git checkout master
git merge feature
```



Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  1. One commit is ancestor of the other

```
git checkout master
git merge feature
```



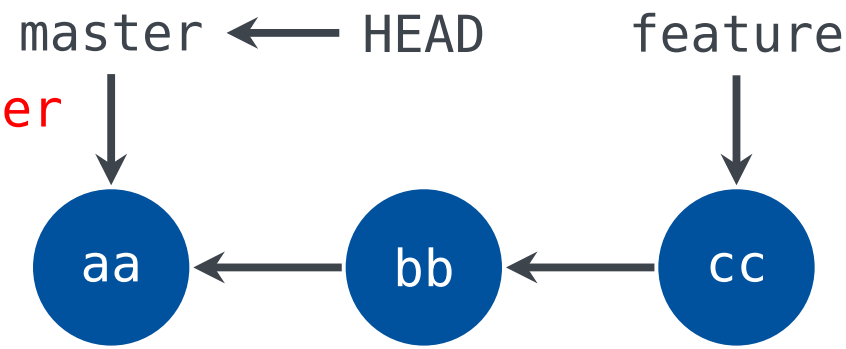
Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  1. One commit is ancestor of the other

Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  1. One commit is ancestor of the other

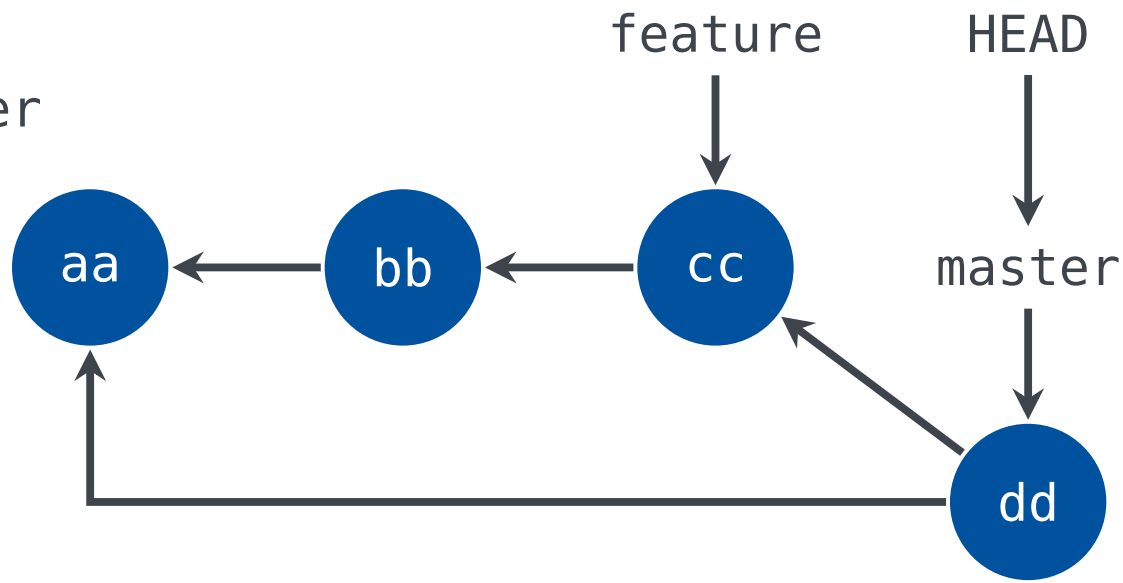
```
git checkout master
git merge --no-ff
  feature
```



Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  1. One commit is ancestor of the other

```
git checkout master
git merge --no-ff
  feature
```



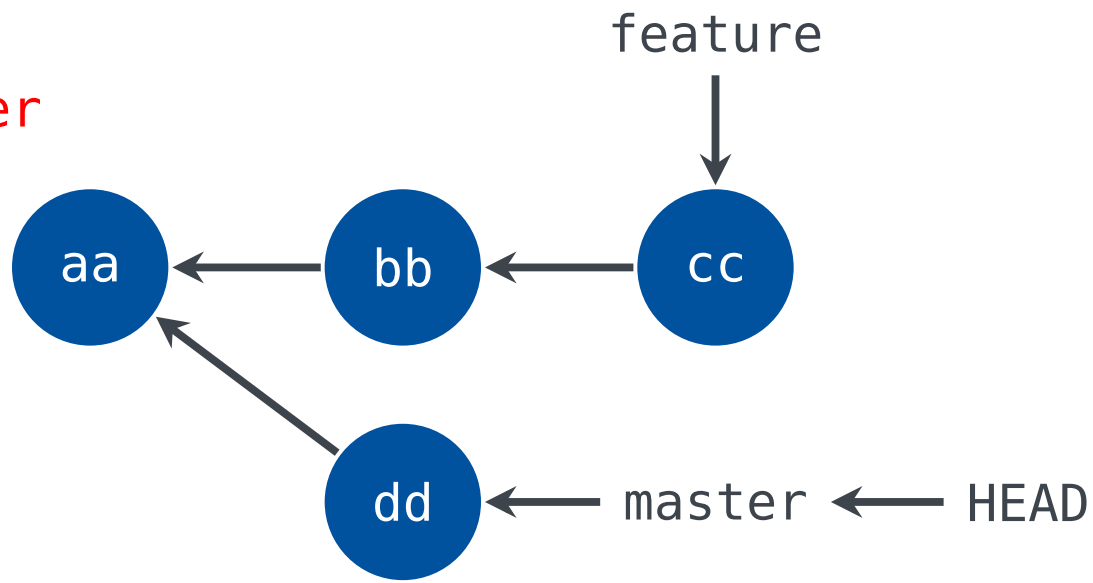
Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  2. Commits have another common ancestor commit

Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  2. Commits have another common ancestor commit

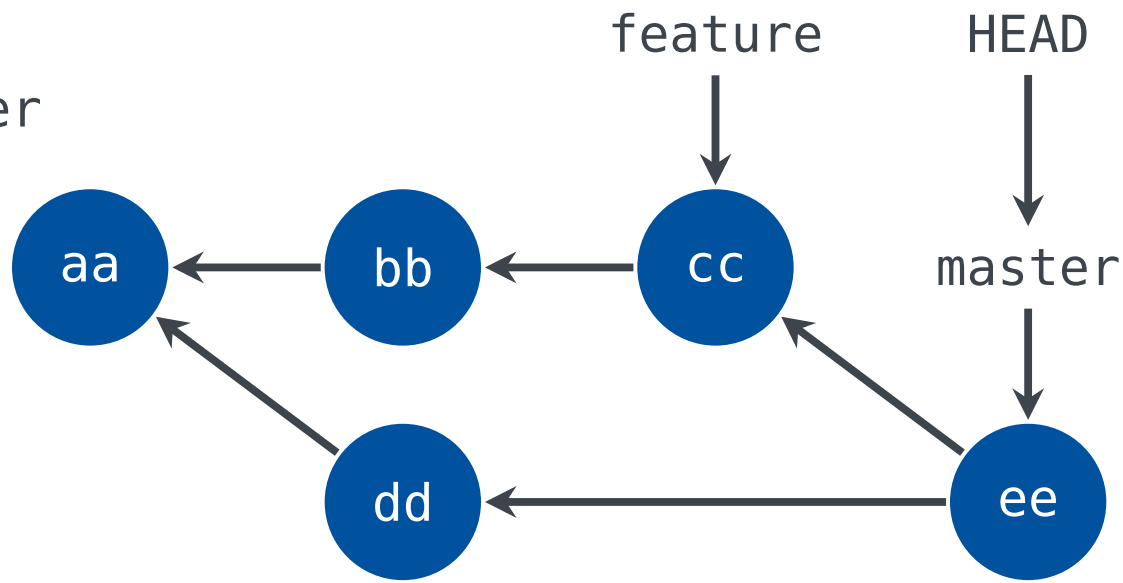
```
git checkout master
git merge feature
```



Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  2. Commits have another common ancestor commit

```
git checkout master
git merge feature
```



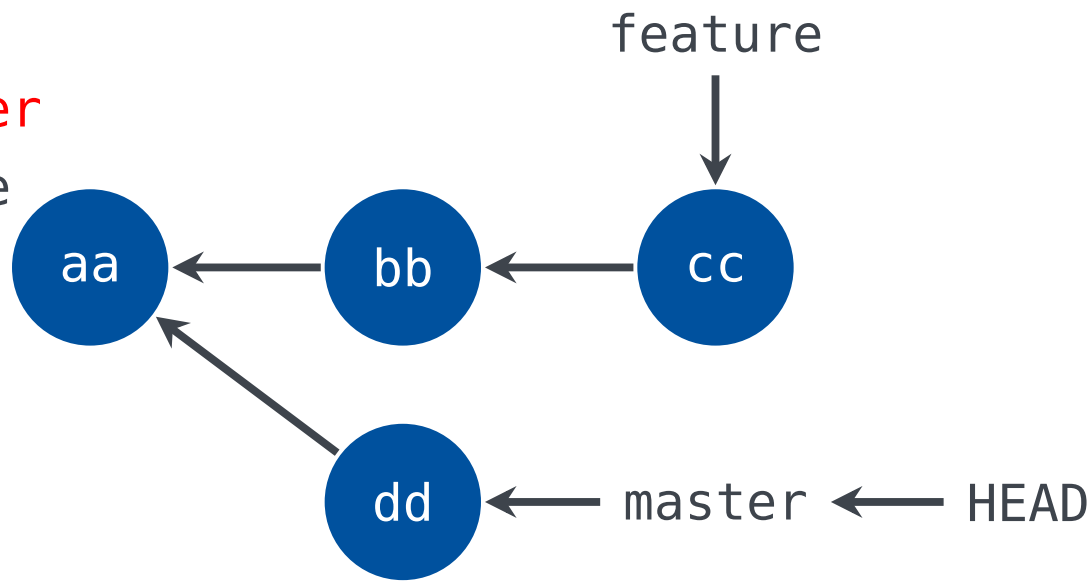
Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  2. Commits have another common ancestor commit

Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  2. Commits have another common ancestor commit

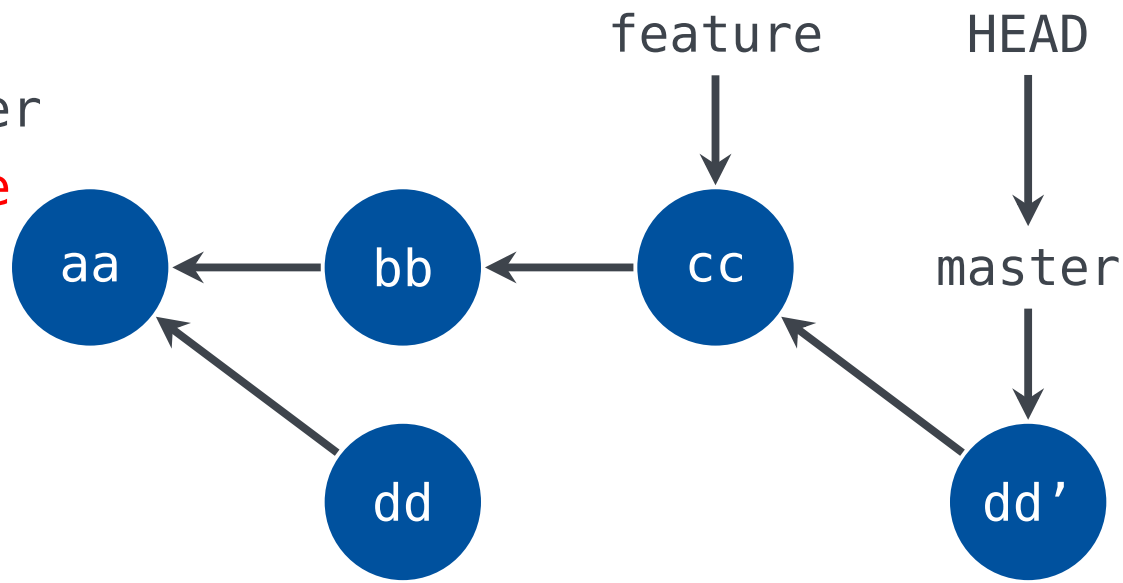
```
git checkout master
git rebase feature
```



Merging

- Different types of merging:
  - **Fast-forward merge:** Move branch to descendant commit
  - **Merge commit:** Create commit with two (or more) parents
  - **Rebasing:** Apply one parent patch after another
- Two cases:
  2. Commits have another common ancestor commit

```
git checkout master
git rebase feature
```



Remotes

- **Git is decentralized:** Every repo is independent of others
- `git fetch`:
  1. Transmit missing commits of remote branch(es)
  2. Create/update remote-tracking branch(es)
- `git pull`:
  1. `git fetch`
  2. `git merge`/`rebase` `FETCH_HEAD`

  Default: Pull upstream counterpart into current branch
- `git push`:
  1. Transmit missing commits of local branch(es)
  2. Create/update remote branch(es)

  Default: Push current branch to its upstream counterpart, non-fast-forward pushes are rejected (override with `--force`)

Remotes

- **Git is decentralized:** Every repo is independent of others
- `git fetch`:
  1. Transmit missing commits of remote branch(es)
  2. Create/update remote-tracking branch(es)
- `git pull`:
  1. `git fetch`
  2. `git merge`/`rebase` `FETCH_HEAD`

  Default: Pull upstream counterpart into current branch
- `git push`:
  1. Transmit missing commits of local branch(es)
  2. Create/update remote branch(es)

  Default: Push current branch to its upstream counterpart, non-fast-forward pushes are rejected (override with `--force`)

Remotes

- **Git is decentralized:** Every repo is independent of others
- `git fetch`:
  1. Transmit missing commits of remote branch(es)
  2. Create/update remote-tracking branch(es)
- `git pull`:
  1. **`git fetch`**
  2. **`git merge`**/**`rebase`** `FETCH_HEAD`

  Default: Pull upstream counterpart into current branch
- `git push`:
  1. Transmit missing commits of local branch(es)
  2. Create/update remote branch(es)

  Default: Push current branch to its upstream counterpart, non-fast-forward pushes are rejected (override with `--force`)

Remotes

- **Git is decentralized:** Every repo is independent of others
- `git fetch`:
  1. Transmit missing commits of remote branch(es)
  2. Create/update remote-tracking branch(es)
- `git pull`:
  1. **`git fetch`**
  2. **`git merge`**/**`rebase`** `FETCH_HEAD`

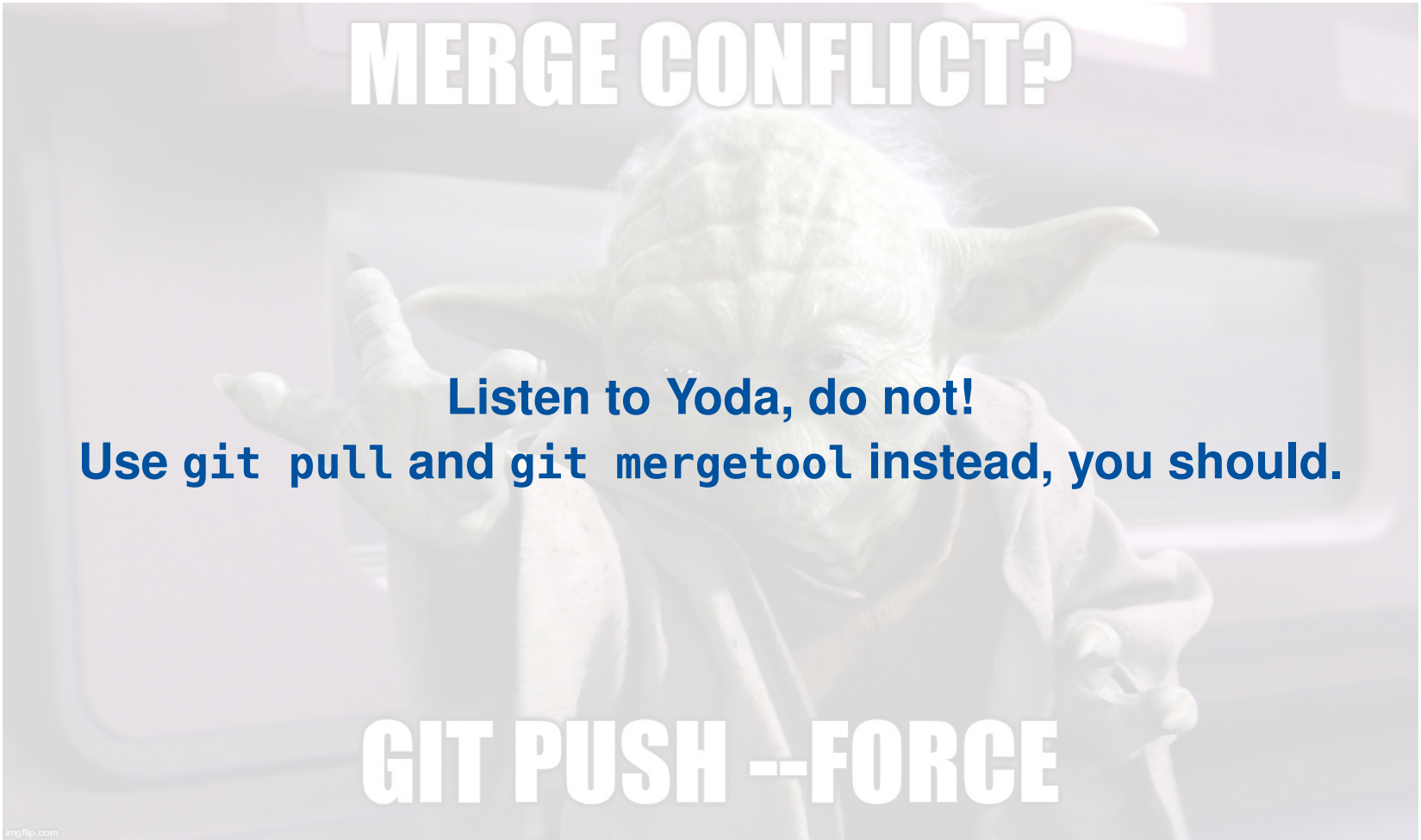
  Default: Pull upstream counterpart into current branch
- `git push`:
  1. Transmit missing commits of local branch(es)
  2. Create/update remote branch(es)

  Default: Push current branch to its upstream counterpart, non-fast-forward pushes are rejected (override with `--force`)

But what should be do when we get merge conflicts?

Use the Force, Luke. (PROTIP: Don't!)

Use the Force, Luke. (PROTIP: Don't!)



Listen to Yoda, do not!

Use `git pull` and `git mergetool` instead, you should.

Comparing Commits


NOBODY CAN GIT BLAME ME
38
imgflip.com

Log

- **`git log:`** List commit history
  - `--stat`/`--shortstat`: Show statistics for modified files
  - `--name-status`: Which files were modified, added, or deleted?
  - `--42`: Only last 42 commits
  - `--before`/`--after`: Only commits before/after date
  - `--author`/`--committer`/`--grep`: Only commits matching a pattern in author/committer/message field
  - `-G`: Only commits where added/deleted lines contain a pattern
  - `-S`: Only commits which change the number of occurrences of a pattern in a file
- **`git diff:`** Show differences between two commits,
  `git difftool:` Use an external tool like Meld
- **`git blame:`** Show for each line of a file the last commit which changed the line

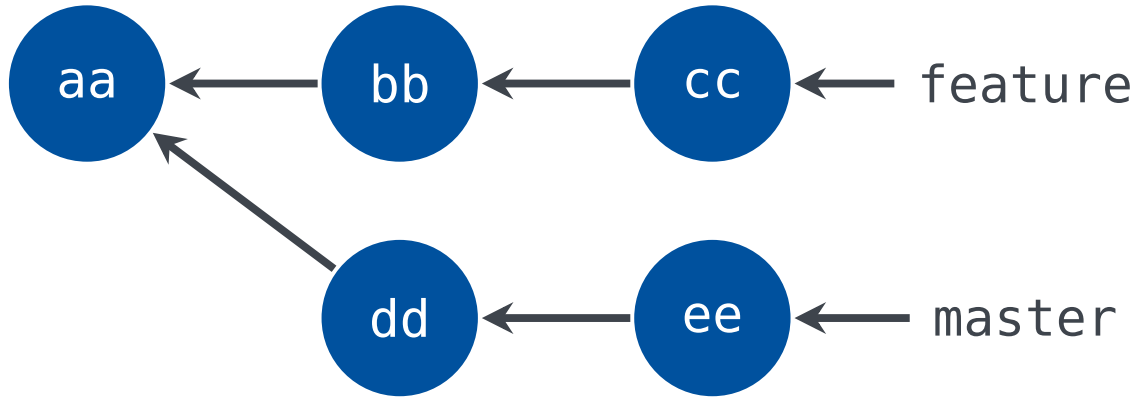
Log Can Also Do Fancy Stuff

```
*   cc24eb Merge remote-tracking branch 'origin/UQ'
|\
| *   9b3cf5 Merge remote-tracking branch 'origin/master' into UQ
| |\
| * | b9b1fe bug fix in dehierarchization of ModPolyBasis -> ...
* | | c92ac9 quasi monte carlo methods removed from master
* | |   e143d1 Merge branch 'master' of https://simsgs.informa...
|\ \ \
| |/ /
|/| /
| |/
| * 711dc8 Temporarily remove shared_ptr wrapping in SWIG
* |   ef252e Merge remote-tracking branch 'origin/master' into UQ
|\ \
| |/
| * 6be35a warnings fixed
| * f87760 sobol sequences removed
* | e6d73d minor changes
|/
*   66ed47 Merge remote-tracking branch 'origin/master' into UQ
```

Commit Navigation

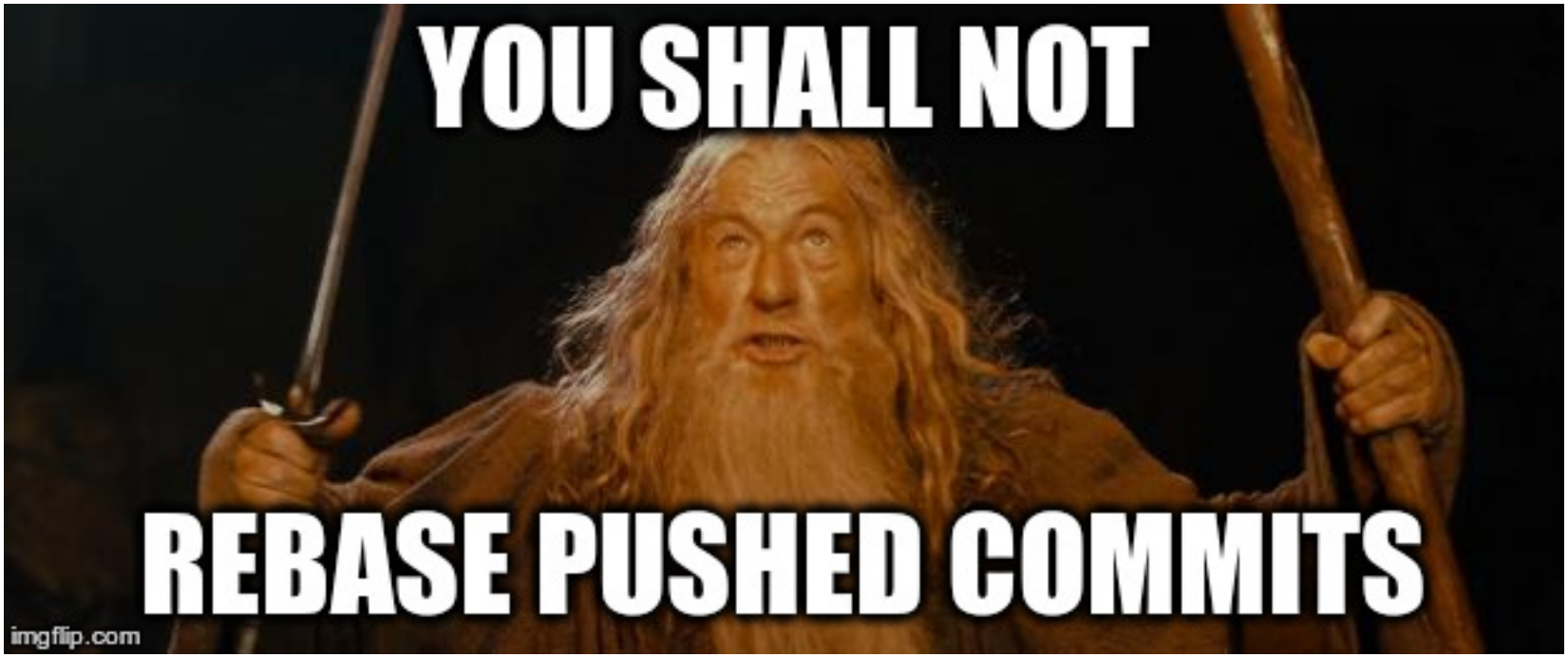
- Many Git commands accept “revisions/ranges“ to specify a commit or a series of commits
- **Single revision:**
  - Short SHA-1: `1c002d` = `1c002dd4b536...`
  - Branch name: `master`
  - Reflog: `HEAD@{5}`, `master@{yesterday}`
  - Parent: `HEAD^`, `HEAD^2` $\neq$ `HEAD^^`
  - First ancestor: `HEAD~`, `HEAD~2` = `HEAD~~`
  - ...

Commit Navigation



- **Commit ranges:**
  - `master..feature`: All commits from the merge base to `feature` (bb, cc)
  - `feature..master`: All commits from the merge base to `master` (dd, ee)
  - `master...feature`: All commits from the merge base to `feature` or `master` (bb, cc, dd, ee)
  - Omit one side to use `HEAD`
- Exception: **`git diff`** compares two *endpoints*
  - **`git diff`** `A..B` = **`git diff`** `A B`
  - **`git diff`** `A...B` = **`git diff`** `$(`**`git merge-base`** `A B) B`
- Type `man gitrevisions` for more info

Rebasing Commits

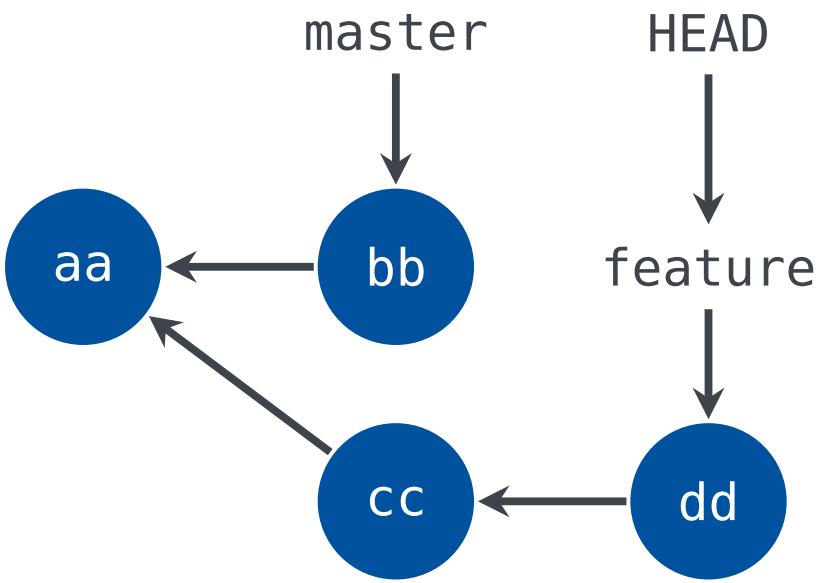

YOU SHALL NOT
REBASE PUSHED COMMITS
imgflip.com

Rebasing

- **`git rebase`:** Create new commits with same changes/patch, but different message/order/parents
- **`git cherry-pick`:** Cherry-pick single commits (bugfixes) on other branch (`release`)
- **Gandalf:** Never rebase commits pushed somewhere else!

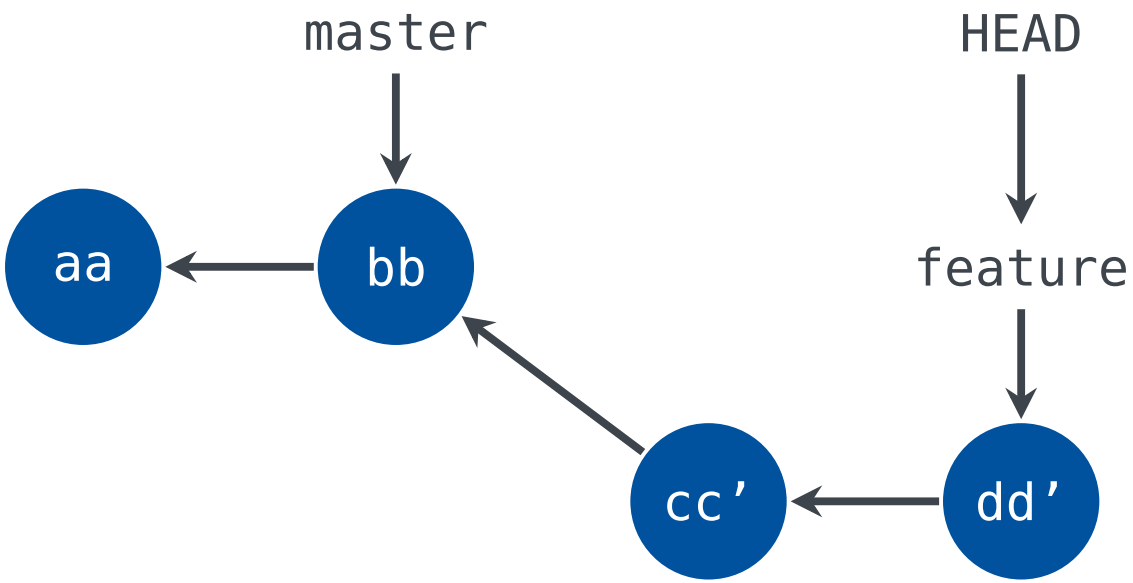
Rebase

```
git checkout feature
git rebase master
```



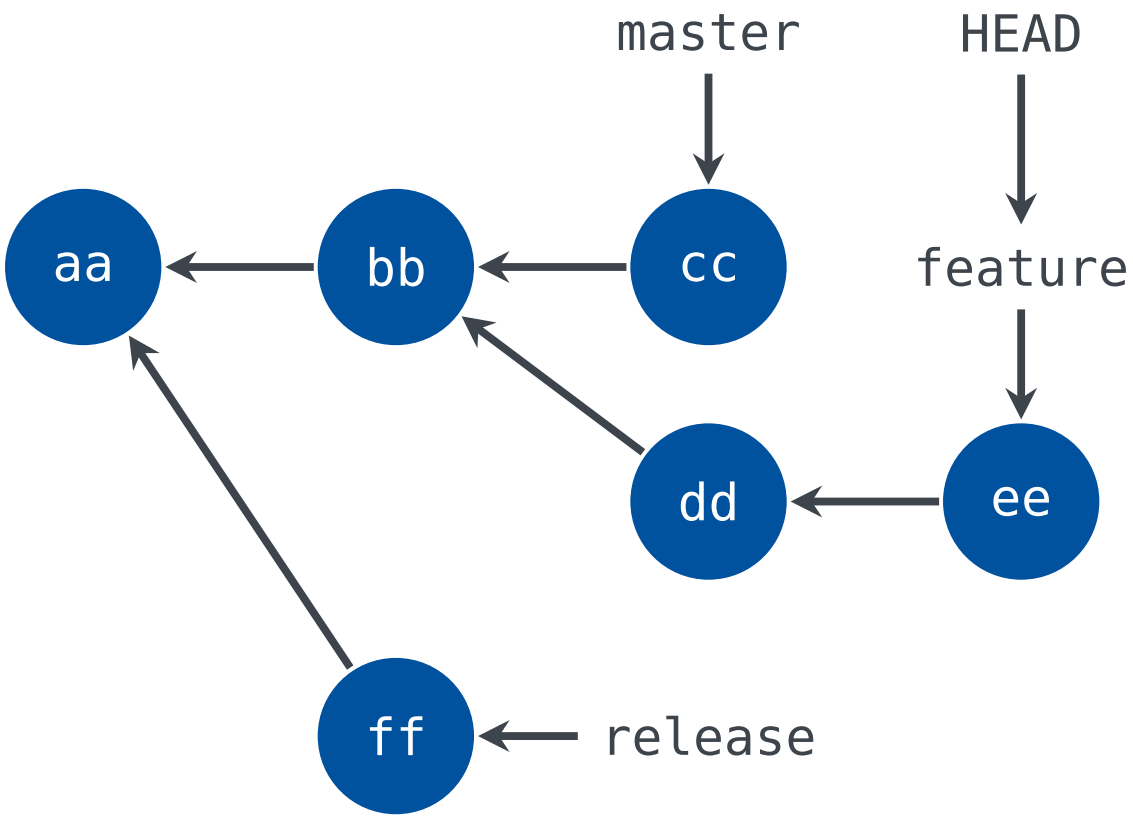
Rebase

```
git checkout feature
git rebase master
```



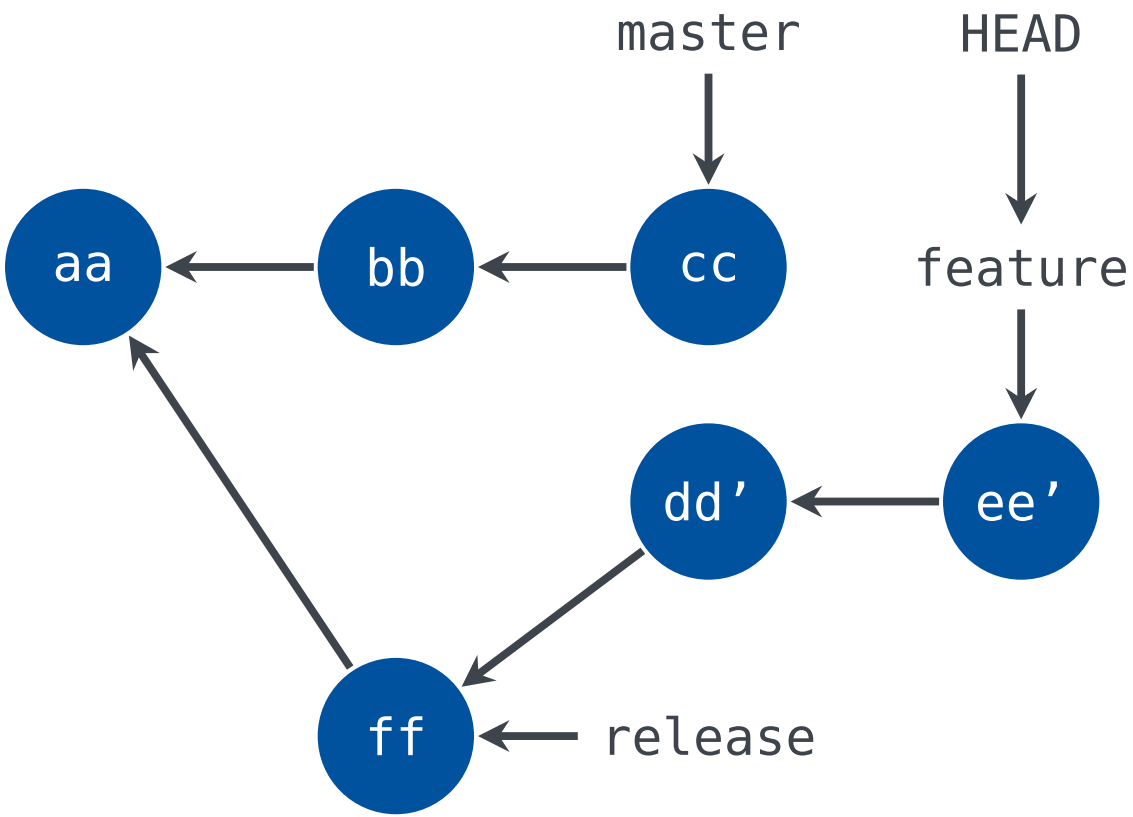
Rebase

```
git checkout feature
git rebase --onto release master
```



Rebase

```
git checkout feature
git rebase --onto release master
```



Interactive Rebase

- **Interactive rebase: `git rebase -i HEAD~3`**

```
pick 58955e5 Fix Doxygen warnings
pick 222bc86 Try to fix renames in non-C++ code
pick 429d079 Fix SWIG Java typemaps in optimization

# Rebase de5b6b4..429d079 onto de5b6b4 (3 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
```

Undoing Stuff

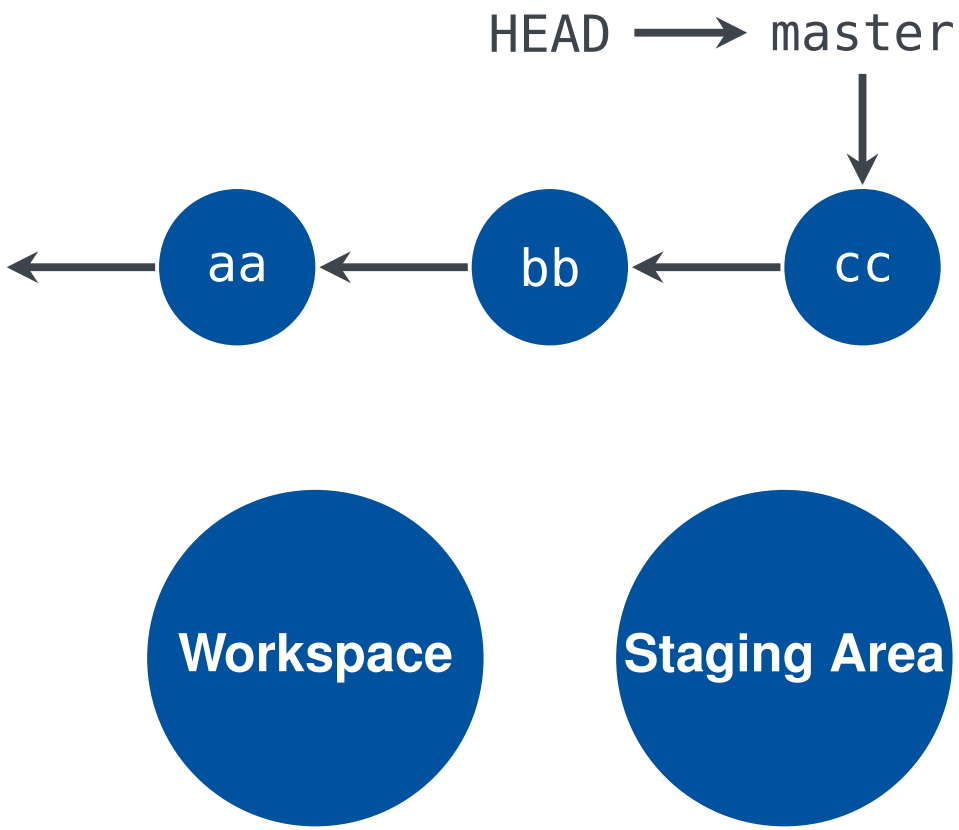

GIT CHECKOUT, GIT RESET, ...
..., GIT RESET --SOFT,
GIT RESET --HARD, ...
WHAT THE HELL
SHOULD I USE?
imgflip.com

Reset and Checkout

- **`git reset`** and **`git checkout`** both change the contents of staging area and/or working directory to other commits.

Initial setting

```
git reset --soft bb
git reset bb
git reset --hard bb
git checkout bb
```



Reset and Checkout

- **`git reset`** and **`git checkout`** both change the contents of staging area and/or working directory to other commits.

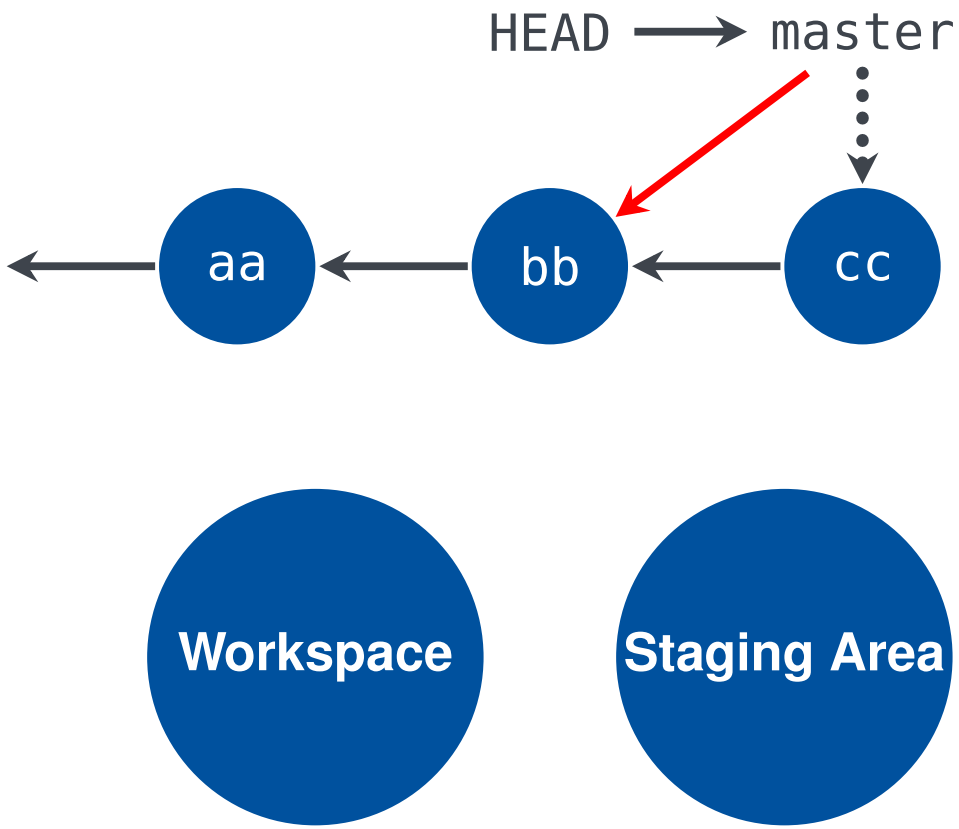
Initial setting

`git reset --soft bb`

`git reset bb`

`git reset --hard bb`

`git checkout bb`



Reset and Checkout

- **`git reset`** and **`git checkout`** both change the contents of staging area and/or working directory to other commits.

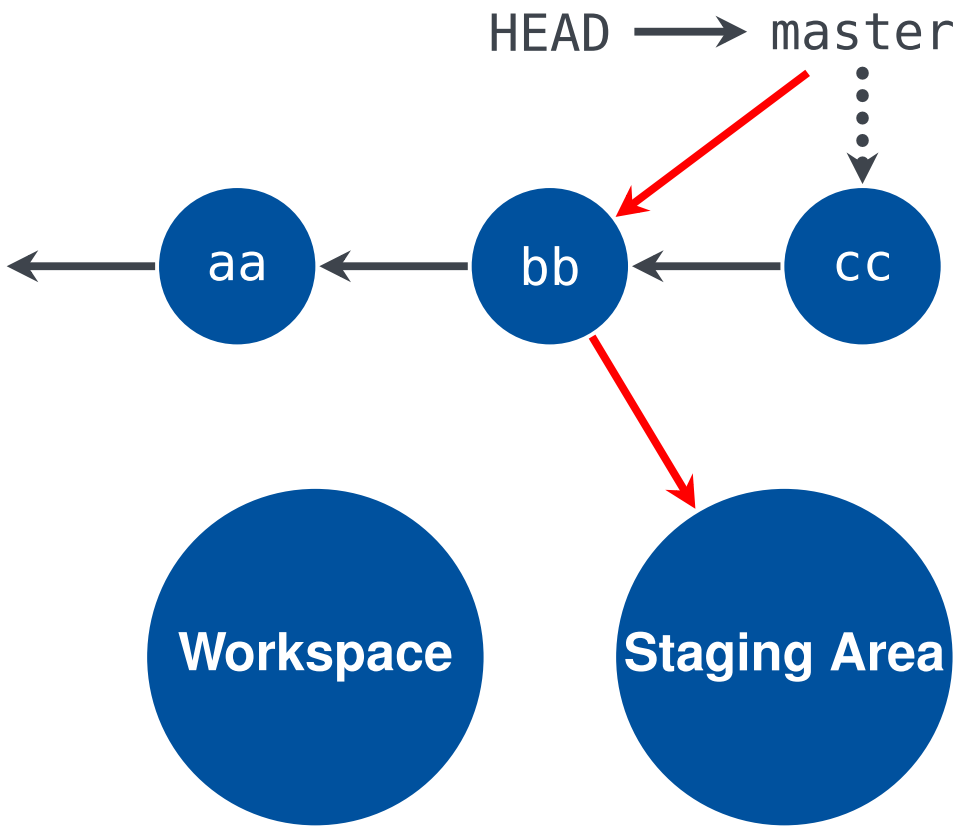
Initial setting

`git reset` `--soft bb`

`git reset` `bb`

`git reset` `--hard bb`

`git checkout` `bb`

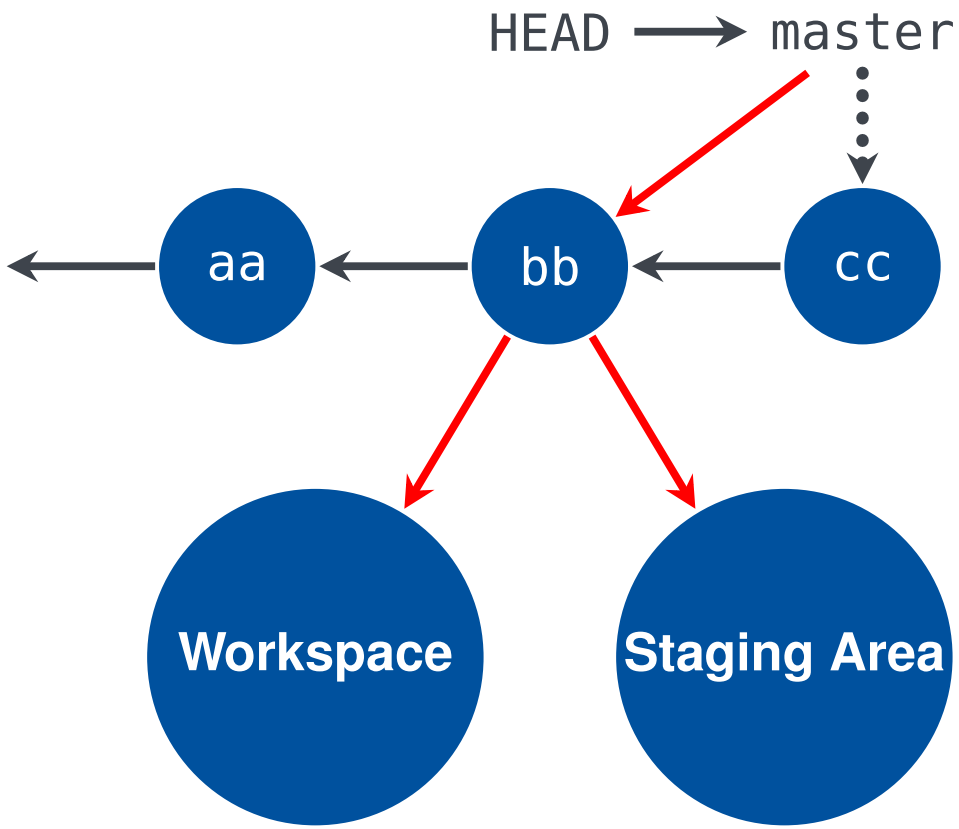


Reset and Checkout

- **`git reset`** and **`git checkout`** both change the contents of staging area and/or working directory to other commits.

Initial setting

```
git reset --soft bb
git reset bb
git reset --hard bb
git checkout bb
```

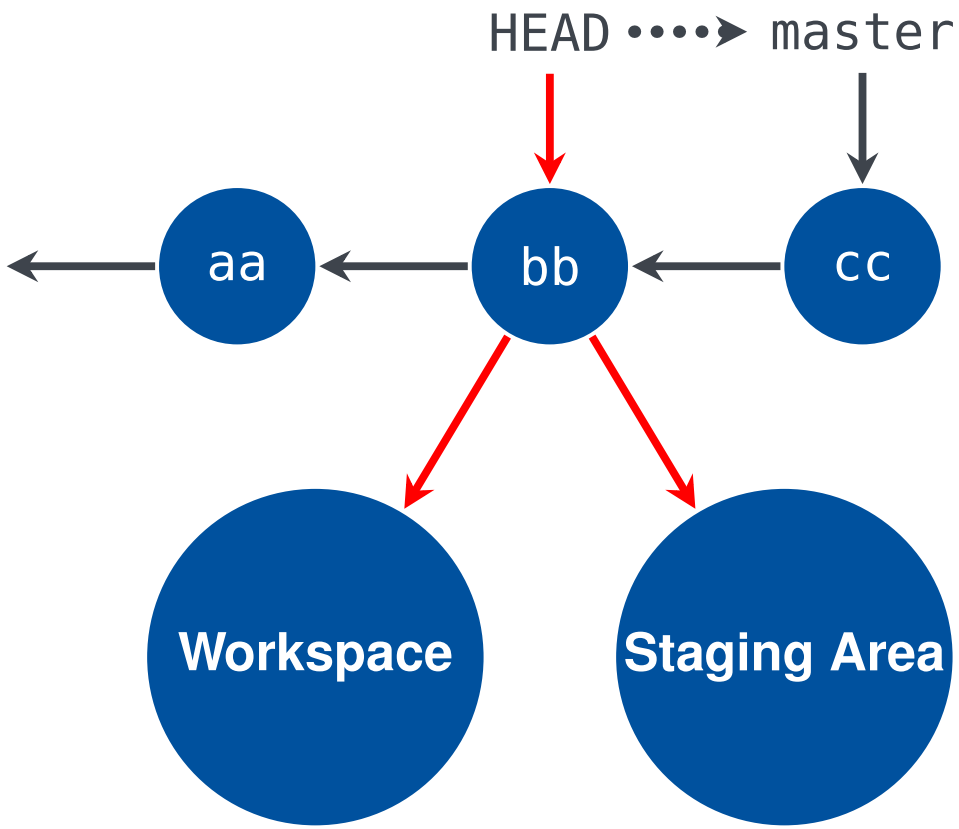


Reset and Checkout

- **`git reset`** and **`git checkout`** both change the contents of staging area and/or working directory to other commits.

Initial setting

```
git reset --soft bb
git reset bb
git reset --hard bb
git checkout bb
```



Reset and Checkout

- **Overview** (from `https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified#Summary`):

Command	Changes	Index	Workspace
Commit Level			
reset `--soft <commit>`	branch	no	no
reset `<commit>`	branch	yes	no
reset `--hard <commit>`	branch	yes	yes
checkout `<commit>`	HEAD	yes	yes
File Level			
reset `<commit> <file>`	–	yes	no
checkout `<commit> <file>`	–	yes	yes

Stashing and Reverting

- **`git stash`:** Temporarily save changes of working directory and staging area
  - `--include-untracked`: Also stash untracked files
  - `--keep-index`: Only stash working directory changes
  - `--patch`: Stash individual hunks of changed files

```
stash@{0}: WIP on sgopt: 382334f Rename HashGridStorage methods
stash@{1}: On sgopt: Change Clang to -Weverything
```

- **`git revert`:** Revert commits already pushed to somewhere else, creating new commits
  - **`git revert`** `HEAD~3`: Revert patch introduced by `HEAD~3`
  - **`git revert`** `HEAD~5..HEAD~2`: Analogously
  - **`git revert`** `--mainline 1 <merge-commit>`: Revert merge commit, choosing the first parent

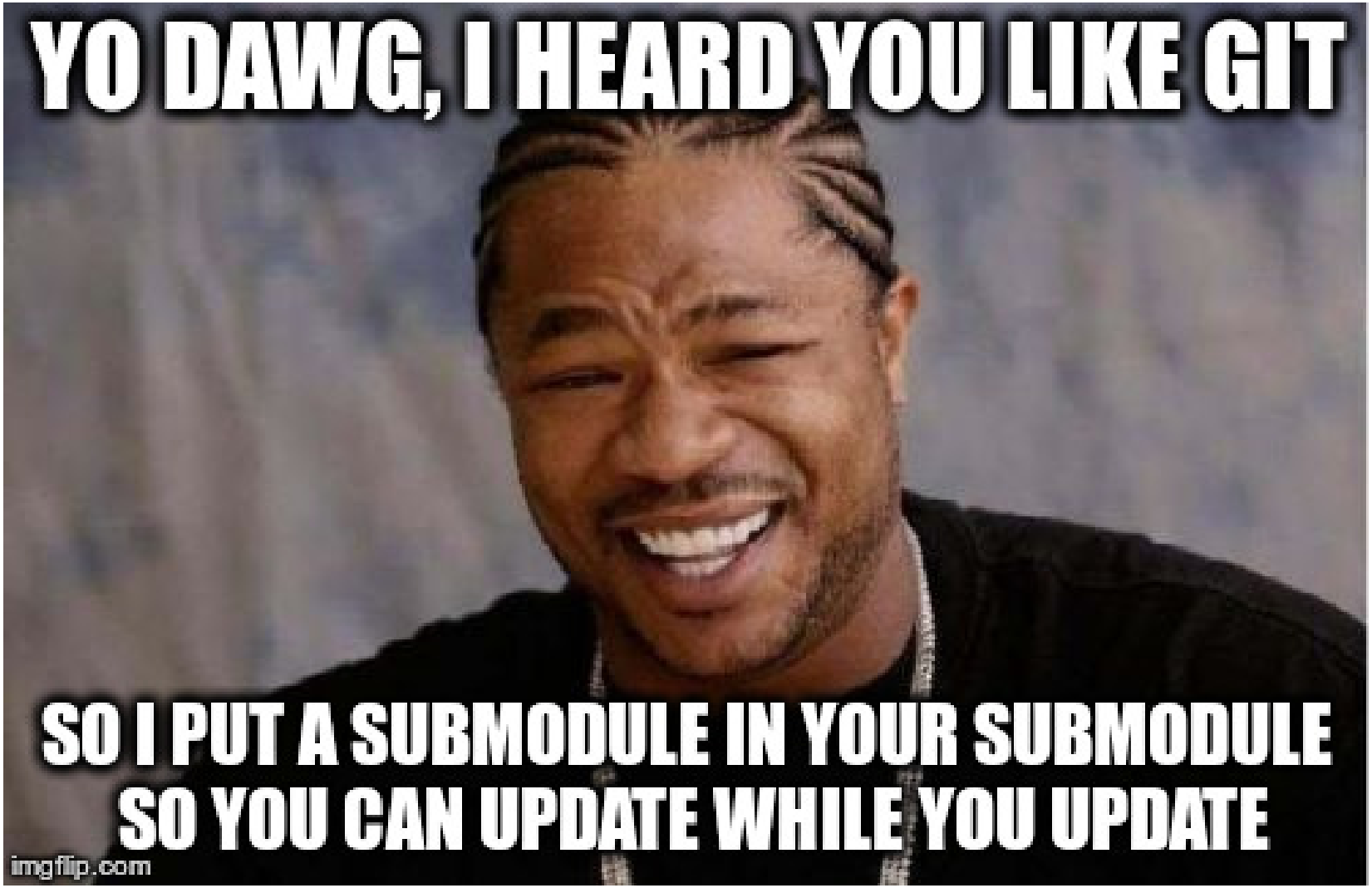
Stashing and Reverting

- **`git stash`:** Temporarily save changes of working directory and staging area
  - `--include-untracked`: Also stash untracked files
  - `--keep-index`: Only stash working directory changes
  - `--patch`: Stash individual hunks of changed files

```
stash@{0}: WIP on sgopt: 382334f Rename HashGridStorage methods
stash@{1}: On sgopt: Change Clang to -Weverything
```

- **`git revert`:** Revert commits already pushed to somewhere else, creating new commits
  - **`git revert`** `HEAD~3`: Revert patch introduced by `HEAD~3`
  - **`git revert`** `HEAD~5..HEAD~2`: Analogously
  - **`git revert`** `--mainline 1 <merge-commit>`: Revert merge commit, choosing the first parent

Last But Not Least Some Miscellaneous Things


YO DAWG, I HEARD YOU LIKE GIT
SO I PUT A SUBMODULE IN YOUR SUBMODULE
SO YOU CAN UPDATE WHILE YOU UPDATE
imgflip.com

Miscellaneous (1/3)

- **Submodules/Subtrees:**
  - **`git submodule`**: Include **reference to commit** of other Git repo in a subdirectory (saved in parent repo: URL, commit hash)
  - **`git subtree`**: Include **complete contents** of other Git repo in a subdirectory (saved in parent repo: contents of all commits)
- **`git bisect`:** Find bugs using bisection when you know initial good/bad commits and can tell if a commit in between is good/bad; can be scripted (**`git bisect`** `run`)
- **`git rerere`:** Automatic merge conflict resolution (**re**use **re**corded **re**solution)
- **`git tag`:** Unmovable names for commits (`release-2.1`)
- **`git filter-branch`:** Rewrite commits automatically (rename/delete files, . . .)

Miscellaneous (1/3)

- **Submodules/Subtrees:**
  - **`git submodule`**: Include **reference to commit** of other Git repo in a subdirectory (saved in parent repo: URL, commit hash)
  - **`git subtree`**: Include **complete contents** of other Git repo in a subdirectory (saved in parent repo: contents of all commits)
- **`git bisect`:** Find bugs using bisection when you know initial good/bad commits and can tell if a commit in between is good/bad; can be scripted (**`git bisect`** `run`)
- **`git rerere`:** Automatic merge conflict resolution (**re**use **re**corded **re**solution)
- **`git tag`:** Unmovable names for commits (`release-2.1`)
- **`git filter-branch`:** Rewrite commits automatically (rename/delete files, . . .)

Miscellaneous (1/3)

- **Submodules/Subtrees:**
  - **`git submodule`**: Include **reference to commit** of other Git repo in a subdirectory (saved in parent repo: URL, commit hash)
  - **`git subtree`**: Include **complete contents** of other Git repo in a subdirectory (saved in parent repo: contents of all commits)
- **`git bisect`:** Find bugs using bisection when you know initial good/bad commits and can tell if a commit in between is good/bad; can be scripted (**`git bisect`** `run`)
- **`git rerere`:** Automatic merge conflict resolution (**re**use **re**corded **re**solution)
- **`git tag`:** Unmovable names for commits (`release-2.1`)
- **`git filter-branch`:** Rewrite commits automatically (rename/delete files, . . .)

Miscellaneous (1/3)

- **Submodules/Subtrees:**
  - **`git submodule`**: Include **reference to commit** of other Git repo in a subdirectory (saved in parent repo: URL, commit hash)
  - **`git subtree`**: Include **complete contents** of other Git repo in a subdirectory (saved in parent repo: contents of all commits)
- **`git bisect`:** Find bugs using bisection when you know initial good/bad commits and can tell if a commit in between is good/bad; can be scripted (**`git bisect`** `run`)
- **`git rerere`:** Automatic merge conflict resolution (**re**use **re**corded **re**solution)
- **`git tag`:** Unmovable names for commits (`release-2.1`)
- **`git filter-branch`:** Rewrite commits automatically (rename/delete files, . . .)

Miscellaneous (1/3)

- **Submodules/Subtrees:**
  - **`git submodule`**: Include **reference to commit** of other Git repo in a subdirectory (saved in parent repo: URL, commit hash)
  - **`git subtree`**: Include **complete contents** of other Git repo in a subdirectory (saved in parent repo: contents of all commits)
- **`git bisect`:** Find bugs using bisection when you know initial good/bad commits and can tell if a commit in between is good/bad; can be scripted (**`git bisect`** `run`)
- **`git rerere`:** Automatic merge conflict resolution (**re**use **re**corded **re**solution)
- **`git tag`:** Unmovable names for commits (`release-2.1`)
- **`git filter-branch`:** Rewrite commits automatically (rename/delete files, . . .)

Miscellaneous (2/3)

- **Configuration:**
  - System (`/etc/gitconfig`), user (`~/.gitconfig`), repo (`<REPO>/.git/config`)
  - Many, many possibilities: `core.editor`/`pager`, `commit.template`, `merge.tool`, `core.autocrlf`, `core.whitespace`, `...`
  - Aliases: Shortcuts for longer commands
- **Attributes:** Config. that only applies to specific paths (e.g., which files are considered binary, how to diff `*.docx` files, ...)
- **`.gitignore`:** Prevent binary files from being added
- **Hooks:**
  - Client-side: `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`, `...`
  - Server-side: `pre-receive`, `update`, `post-receive`, `...`

Miscellaneous (2/3)

- **Configuration:**
  - System (`/etc/gitconfig`), user (`~/.gitconfig`), repo (`<REPO>/.git/config`)
  - Many, many possibilities: `core.editor`/`pager`, `commit.template`, `merge.tool`, `core.autocrlf`, `core.whitespace`, `...`
  - Aliases: Shortcuts for longer commands
- **Attributes:** Config. that only applies to specific paths (e.g., which files are considered binary, how to diff `*.docx` files, . . .)
- **`.gitignore`:** Prevent binary files from being added
- **Hooks:**
  - Client-side: `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`, `...`
  - Server-side: `pre-receive`, `update`, `post-receive`, `...`

Miscellaneous (2/3)

- **Configuration:**
  - System (`/etc/gitconfig`), user (`~/.gitconfig`), repo (`<REPO>/.git/config`)
  - Many, many possibilities: `core.editor`/`pager`, `commit.template`, `merge.tool`, `core.autocrlf`, `core.whitespace`, `...`
  - Aliases: Shortcuts for longer commands
- **Attributes:** Config. that only applies to specific paths (e.g., which files are considered binary, how to diff `*.docx` files, ...)
- **`.gitignore`:** Prevent binary files from being added
- **Hooks:**
  - Client-side: `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`, `...`
  - Server-side: `pre-receive`, `update`, `post-receive`, `...`

Miscellaneous (2/3)

- **Configuration:**
  - System (`/etc/gitconfig`), user (`~/.gitconfig`), repo (`<REPO>/.git/config`)
  - Many, many possibilities: `core.editor`/`pager`, `commit.template`, `merge.tool`, `core.autocrlf`, `core.whitespace`, `...`
  - Aliases: Shortcuts for longer commands
- **Attributes:** Config. that only applies to specific paths (e.g., which files are considered binary, how to diff `*.docx` files, ...)
- **`.gitignore`:** Prevent binary files from being added
- **Hooks:**
  - Client-side: `pre-commit`, `prepare-commit-msg`, `commit-msg`, `post-commit`, `...`
  - Server-side: `pre-receive`, `update`, `post-receive`, `...`

Miscellaneous (3/3)

- **`git reflog`:** Show previous positions of `HEAD`

```
f8b42cf HEAD@{0}: checkout: moving from master to sgopt
fc7f1ed HEAD@{1}: rebase finished: returning to refs/heads/master
fc7f1ed HEAD@{2}: pull: checkout fc7f1ed23d0ec1e02e62538c9ad7fb43
f8b42cf HEAD@{3}: checkout: moving from optUQ to master
fcb9adc HEAD@{4}: commit: Fix typo in OperationQuadrature
```

- **`git gc`:** Garbage collect, remove unreachable blobs/commits
- **Porcelain/Plumbing:**
  - Porcelain: User-friendly commands (**`branch`**, **`checkout`**, ...)
  - Plumbing: Low-level commands (**`ls-tree`**, **`hash-object`**, ...)
- **Getting help:**
  - `man git <COMMAND>` (`<COMMAND>`-specific help)
  - `man gittutorial`, `man gittutorial-2` (tutorials)
  - `man giteveryday` (set of essential Git commands)
  - `man gitglossary` (glossary of Git words)

Miscellaneous (3/3)

- **`git reflog`:** Show previous positions of `HEAD`

```
f8b42cf HEAD@{0}: checkout: moving from master to sgopt
fc7f1ed HEAD@{1}: rebase finished: returning to refs/heads/master
fc7f1ed HEAD@{2}: pull: checkout fc7f1ed23d0ec1e02e62538c9ad7fb43
f8b42cf HEAD@{3}: checkout: moving from optUQ to master
fcb9adc HEAD@{4}: commit: Fix typo in OperationQuadrature
```

- **`git gc`:** Garbage collect, remove unreachable blobs/commits
- **Porcelain/Plumbing:**
  - Porcelain: User-friendly commands (**`branch`**, **`checkout`**, ...)
  - Plumbing: Low-level commands (**`ls-tree`**, **`hash-object`**, ...)
- **Getting help:**
  - `man git <COMMAND>` (`<COMMAND>`-specific help)
  - `man gittutorial`, `man gittutorial-2` (tutorials)
  - `man giteveryday` (set of essential Git commands)
  - `man gitglossary` (glossary of Git words)

Miscellaneous (3/3)

- **`git reflog`:** Show previous positions of `HEAD`

```
f8b42cf HEAD@{0}: checkout: moving from master to sgopt
fc7f1ed HEAD@{1}: rebase finished: returning to refs/heads/master
fc7f1ed HEAD@{2}: pull: checkout fc7f1ed23d0ec1e02e62538c9ad7fb43
f8b42cf HEAD@{3}: checkout: moving from optUQ to master
fcb9adc HEAD@{4}: commit: Fix typo in OperationQuadrature
```

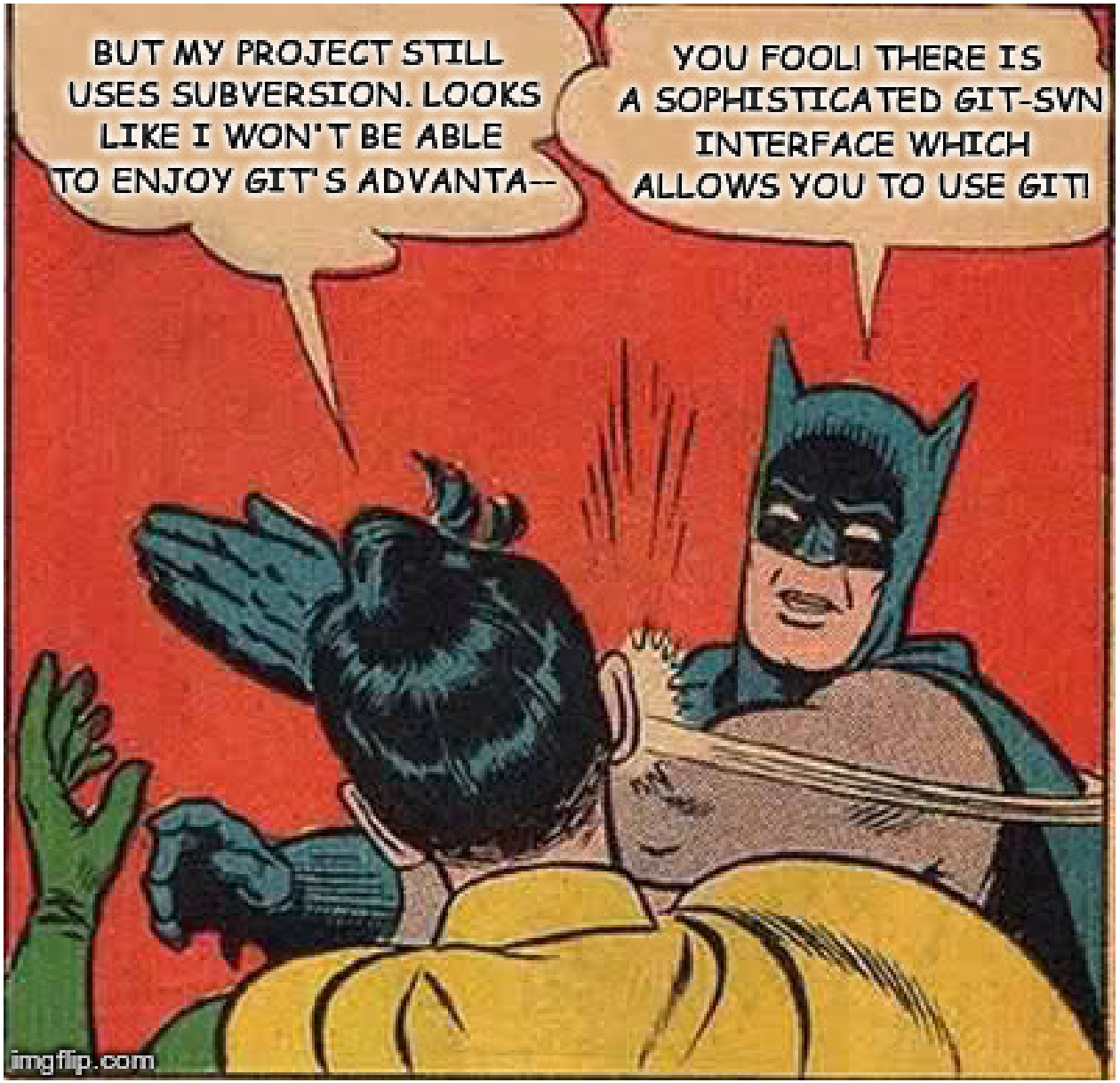
- **`git gc`:** Garbage collect, remove unreachable blobs/commits
- **Porcelain/Plumbing:**
  - Porcelain: User-friendly commands (**`branch`**, **`checkout`**, ...)
  - Plumbing: Low-level commands (**`ls-tree`**, **`hash-object`**, ...)
- **Getting help:**
  - `man git <COMMAND>` (`<COMMAND>`-specific help)
  - `man gittutorial`, `man gittutorial-2` (tutorials)
  - `man giteveryday` (set of essential Git commands)
  - `man gitglossary` (glossary of Git words)

Miscellaneous (3/3)

- **`git reflog`:** Show previous positions of `HEAD`

```
f8b42cf HEAD@{0}: checkout: moving from master to sgopt
fc7f1ed HEAD@{1}: rebase finished: returning to refs/heads/master
fc7f1ed HEAD@{2}: pull: checkout fc7f1ed23d0ec1e02e62538c9ad7fb43
f8b42cf HEAD@{3}: checkout: moving from optUQ to master
fcb9adc HEAD@{4}: commit: Fix typo in OperationQuadrature
```

- **`git gc`:** Garbage collect, remove unreachable blobs/commits
- **Porcelain/Plumbing:**
  - Porcelain: User-friendly commands (**`branch`**, **`checkout`**, ...)
  - Plumbing: Low-level commands (**`ls-tree`**, **`hash-object`**, ...)
- **Getting help:**
  - `man git <COMMAND>` (<COMMAND>-specific help)
  - `man gittutorial`, `man gittutorial-2` (tutorials)
  - `man giteveryday` (set of essential Git commands)
  - `man gitglossary` (glossary of Git words)


BUT MY PROJECT STILL USES SUBVERSION. LOOKS LIKE I WON'T BE ABLE TO ENJOY GIT'S ADVANTA--
YOU FOOL! THERE IS A SOPHISTICATED GIT-SVN INTERFACE WHICH ALLOWS YOU TO USE GIT!
imgflip.com

Thank you for your attention!

IPVS

SGS

SimTech
Cluster of Excellence

Literature

- S. Chacon and B. Straub. *Pro Git*. 2nd ed. Licensed under CC BY-NC-SA 3.0 Unported. 2016. URL: https://git-scm.com/book/en/v2
- R. Preißel and B. Stachmann. *Git. Dezentrale Versionsverwaltung im Team – Grundlagen und Workflows*. 3rd ed. Heidelberg: dpunkt.verlag, 2016