

An introduction of plumbing and porcelain commands



git

Welcome!



Plumbing and Porcelain

Git Commands: Plumbing and Porcelain

plumbing

plumbing consists of low-level commands that enable basic content tracking and the manipulation of directed acyclic graphs (DAG)

porcelain

smaller subset of git commands that most Git end users are likely to need to use for maintaining repositories and communicating between repositories for collaboration

porcelain

git-add	git-rebase	git-cherry
git-am	git-reset	git-count-objects
git-archive	git-revert	git-difftool
git-bisect	git-rm	git-fsck
git-branch	git-shortlog	git-get-tar-commit-id
git-bundle	git-show	git-help
git-checkout	git-stash	git-instaweb
git-cherry-pick	git-status	git-merge-tree
git-citool	git-submodule	git-rerere
git-clean	git-tag	git-rev-parse
git-clone	git-worktree	git-show-branch
git-commit	gitk	git-verify-commit
git-describe	git-config	git-verify-tag
git-diff	git-fast-export	git-whatchanged
git-fetch	git-fast-import	gitweb
git-format-patch	git-filter-branch	git-archimport
git-gc	git-mergetool	git-cvsexportcommit
git-grep	git-pack-refs	git-cvsimport
git-gui	git-prune	git-cvsserver
git-init	git-reflog	git-imap-send
git-log	git-relink	git-p4
git-merge	git-remote	git-quiltimport
git-mv	git-repack	git-request-pull
git-notes	git-replace	git-send-email
git-pull	git-annotate	git-svn
git-push	git-blame	

plumbing

git-apply	git-for-each-ref	git-receive-pack
git-checkout-index	git-ls-files	git-shell
git-commit-tree	git-ls-remote	git-upload-archive
git-hash-object	git-ls-tree	git-upload-pack
git-index-pack	git-merge-base	git-check-attr
git-merge-file	git-name-rev	git-check-ignore
git-merge-index	git-pack-redundant	git-check-mailmap
git-mktag	git-rev-list	git-check-ref-format
git-mktree	git-show-index	git-column
git-pack-objects	git-show-ref	git-credential
git-prune-packed	git-unpack-file	git-credential-cache
git-read-tree	git-var	git-credential-store
git-symbolic-ref	git-verify-pack	git-fmt-merge-msg
git-unpack-objects	git-daemon	git-interpret-trailers
git-update-index	git-fetch-pack	git-mailinfo
git-update-ref	git-http-backend	git-mailsplit
git-write-tree	git-send-pack	git-merge-one-file
git-cat-file	git-update-server-info	git-patch-id
git-diff-files	git-http-fetch	git-sh-i18n
git-diff-index	git-http-push	git-sh-setup
git-diff-tree	git-parse-remote	git-stripspace

Low Level Commands (Plumbing)

cat-file, commit-tree, count-objects, diff-index, for-each-ref, hash-object, merge-base, read-tree, rev-list, rev-parse, show-ref, symbolic-ref, update-index, update-ref, verify-pack, write-tree

High Level Commands (Porcelain)

`commit/push/pull/merge/...`

- meant to be readable by human
- not meant to be parsed
- susceptible to changes/evolutions

Describing Changeset with Hash Value

Pros

- Same content never duplicate

Cons

- Managing multiple files?
- Managing history?
- Hard to understand hash values

Vernaculars and Plumbing Commands

Git Vernaculars

blob

stores file content.

tree

stores directory layouts and filenames.

commit

stores commit info and forms the Git commit graph.

tag

stores annotated tag

Git: A Quick Glance

- Content addressable database
- Key is SHA-1 hash of object's content
- Value is the content
- Same content never duplicate

Setup

```
$ git init demo; cd demo
```

```
Initialized empty Git repository in demo/.git/
```

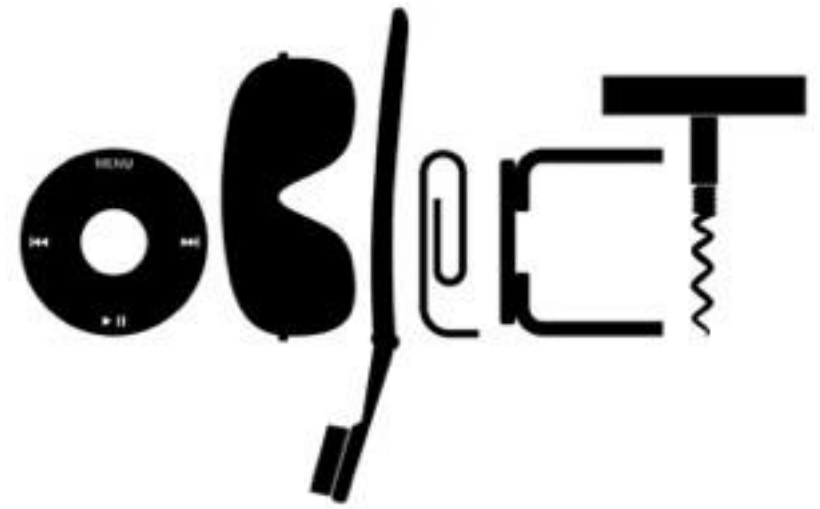
```
# monitoring .git folder
```

```
# if you are using bash, then...
```

```
$ while ;; do tree .git; sleep 2; done
```

object in Git

Same content never duplicate



Parsing Object

```
$ echo "test content" | git hash-object -w --stdin  
d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ find .git/objects/ -type f  
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ git cat-file -p d670  
test content
```

```
$ git cat-file -t d670  
blob
```

hash-object in Git

```
content    = "test content"
header    = "blob %d\0", length_of(content)
store     = header + content
sha1      = sha1_of(store)
dir       = ".git/objects/" + sha1[0:2] + "/"
filename  = sha1[2:]

write(dir + filename, store)
# Save compressed header + content
```


hash-object in Git

```
$ echo "Simon" > test.txt
```

```
$ git hash-object -w test.txt  
83fba84900292b2feb7e955c72eda04a1faa127  
3
```

```
$ echo "Garfunke1" > test.txt
```

```
$ git hash-object -w test.txt  
6cc4930db0f5ccd17f0bc7bfe0ae850b1b0c06a  
2
```

```
$ git cat-file -p 83fb  
Simon
```

```
cat test.txt  
Garfunke1
```

tree Object

- point to other objects (hashed) with name
- a root tree Object is a snapshot



tree Object

```
$ mkdir favorites; echo "fantastic" > favorites/band
```

```
$ git update-index --add test.txt favorites/band
```

```
$ git write-tree
```

```
d0ef546493e53c3beb98e5da4f437e581ff3c7e0
```

```
$ git cat-file -p d0ef
```

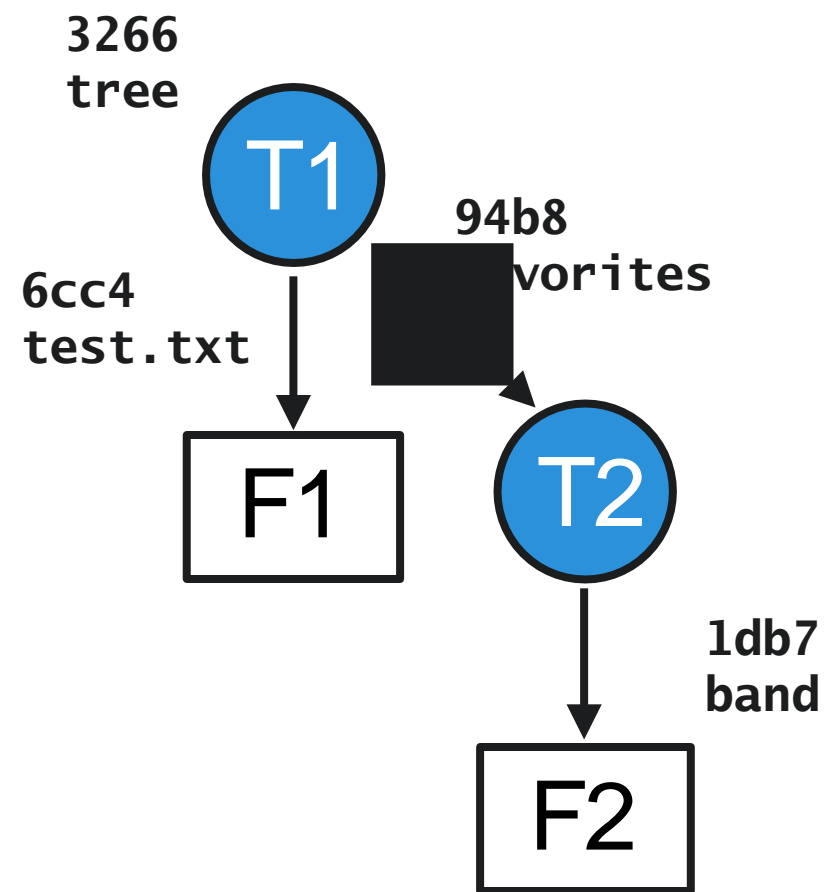
```
040000 tree 94b88a81db61ed617621138bc172aa6f3a408545  
favorites
```

```
100644 blob 6cc4930db0f5ccd17f0bc7bfe0ae850b1b0c06a2  
test.txt
```

```
$ git cat-file -p 94b8
```

```
100644 blob 744c44c7d1ffb9be8db8cd9f1e2c47830ca6f10b band
```

Data Structure



tree Object

```
$ echo "Byrds" > test.txt
```

```
$ git update-index --add test.txt
```

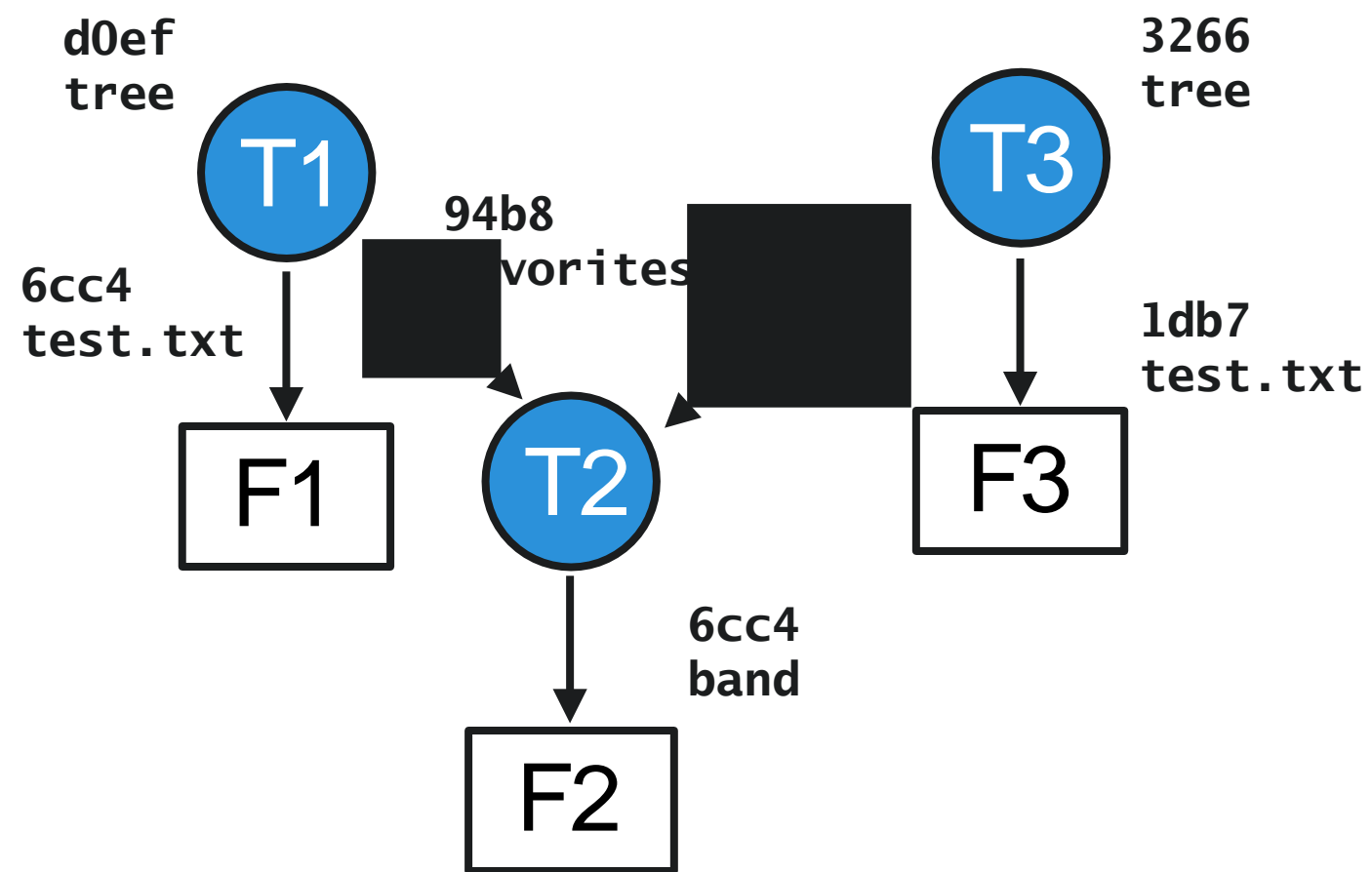
```
$ git write-tree
```

```
3266237ac900a5f09c266b7d6bc79bde6a2386d  
f
```

```
040000 tree 94b88a81db61ed617621138bc172aa6f3a40854  
040000 file 1db792a2c726b393332a882fe105bc3ade4ddb6  
favorites 5  
100644 blob 1db792a2c726b393332a882fe105bc3ade4ddb6  
test.txt 6
```

```
$ git cat-file -p 1db7  
Byrds
```

Data Structure



commit Object

- explain a change: who/when/why
- point to a tree object with above info
- pointer, not delta

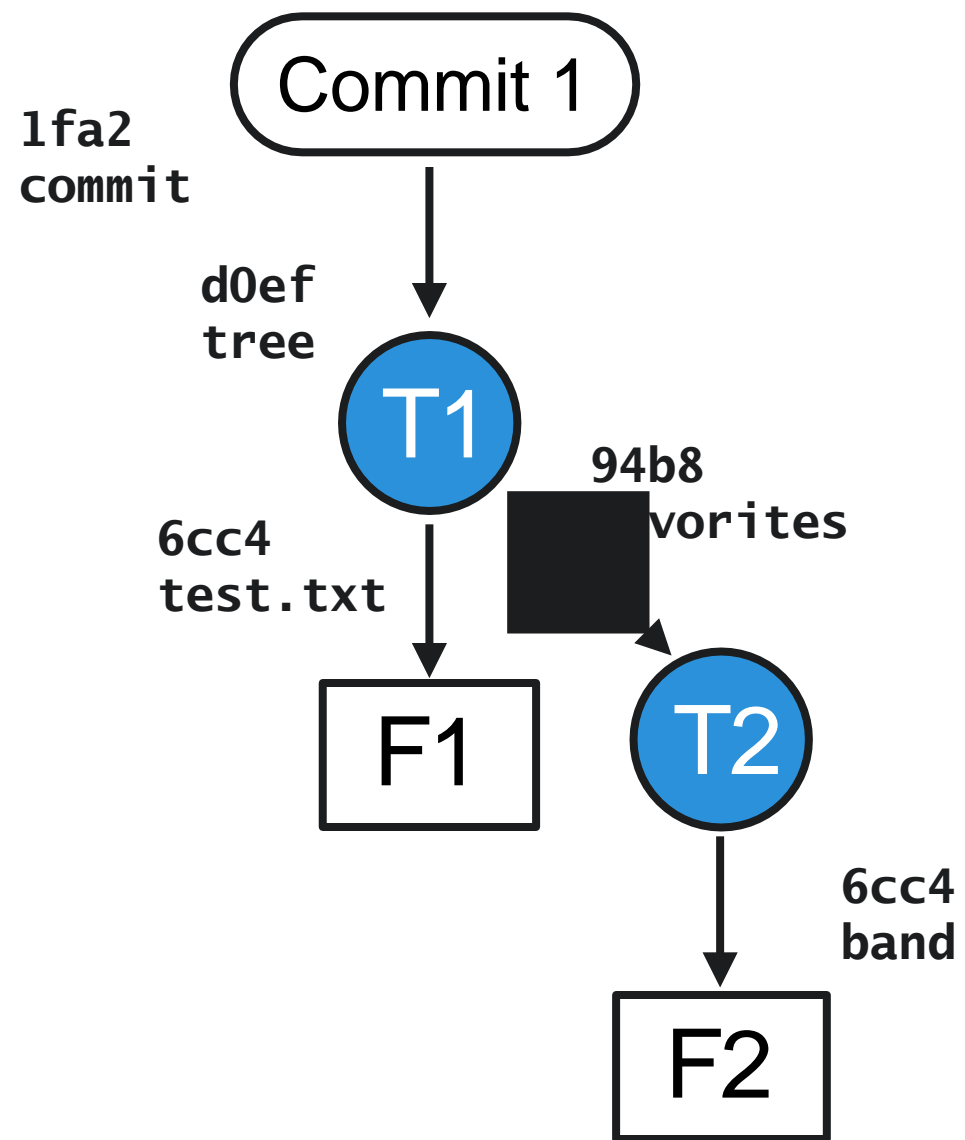
commit Object

```
$ echo "test commit" | git commit-tree d0ef  
1fa215b14c27f2739515eb8410f06478eb0c423e
```

```
$ git cat-file -p 1fa2  
tree d0ef546493e53c3beb98e5da4f437e581ff3c7e0  
author Jingwei "John" Liu <liujingwei@gmail.com>  
1407123580 +0800  
committer Jingwei "John" Liu <liujingwei@gmail.com>  
1407123580 +0800
```

```
test commit
```


Data Structure



commit Object

```
$ echo "another test commit" | git commit-tree 3266 -p  
1fa2  
ded15c0a38dcf226cfbc4838a9d78cd3f8d82baf
```

```
$ git cat-file -p ded1  
tree 3266237ac900a5f09c266b7d6bc79bde6a2386df  
parent 1fa215b14c27f2739515eb8410f06478eb0c423e  
author Jingwei "John" Liu <liujingwei@gmail.com>  
1407123885 +0800  
committer Jingwei "John" Liu <liujingwei@gmail.com>  
1407123885 +0800
```

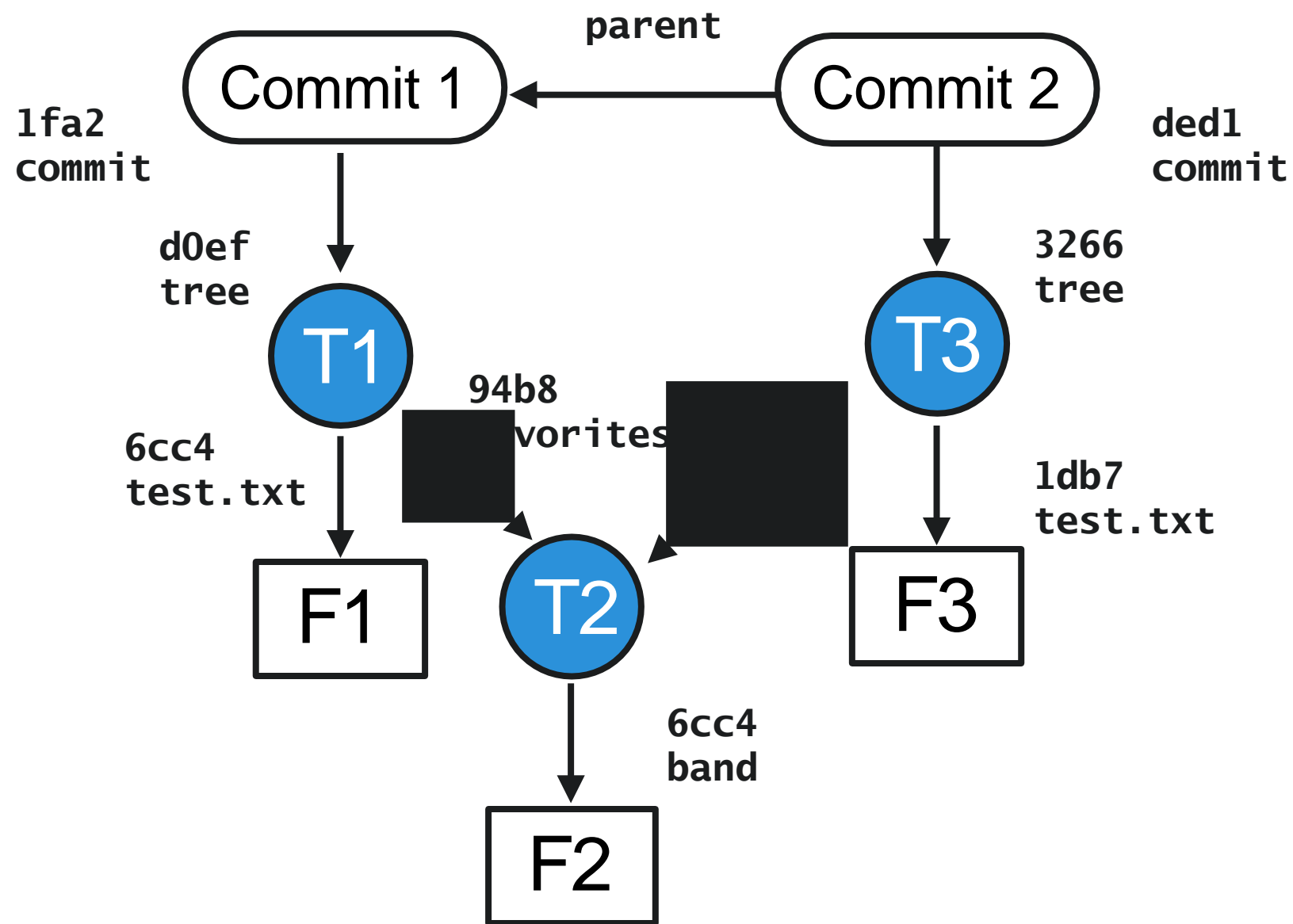
another test commit

commit Object Anatomy

```
$ git cat-file -p ded1
tree [SHA-1]
parent [SHA-1]
author [name] [email] [timestamp] [timezone]
committer [name] [email] [timestamp] [timezone]

[commit message]
```

Data Structure



commit Object

- contains metadata about its ancestors
- can have zero or many (theoretically unlimited) parent commits
- initial commit has no parent
- three-way merge commit has three parents

reference in Git

- stored in .git/refs
- SHA-1

reference in Git

```
# create a branch
```

```
$ echo "1fa215b14c27f2739515eb8410f06478eb0c423e"
```

```
> .git/refs/heads/test-branch
```

```
$ git branch
```

```
* master
```

```
test-branch
```

```
$ git log --pretty=oneline test-branch
```

```
1fa215b14c27f2739515eb8410f06478eb0c423e test commit
```

```
$ find .git/refs/heads -type f
```

```
.git/refs/heads/master
```

```
.git/refs/heads/test-branch
```

reference in Git

```
$ git update-refs refs/heads/master  
ded15c0a38dcf226cfbc4838a9d78cd3f8d82baf
```

```
$git log --pretty=oneline master  
ded15c0a38dcf226cfbc4838a9d78cd3f8d82baf another test  
commit  
1fa215b14c27f2739515eb8410f06478eb0c423e test commit
```

```
$ cat .git/refs/heads/master  
ded15c0a38dcf226cfbc4838a9d78cd3f8d82baf
```


reference in Git

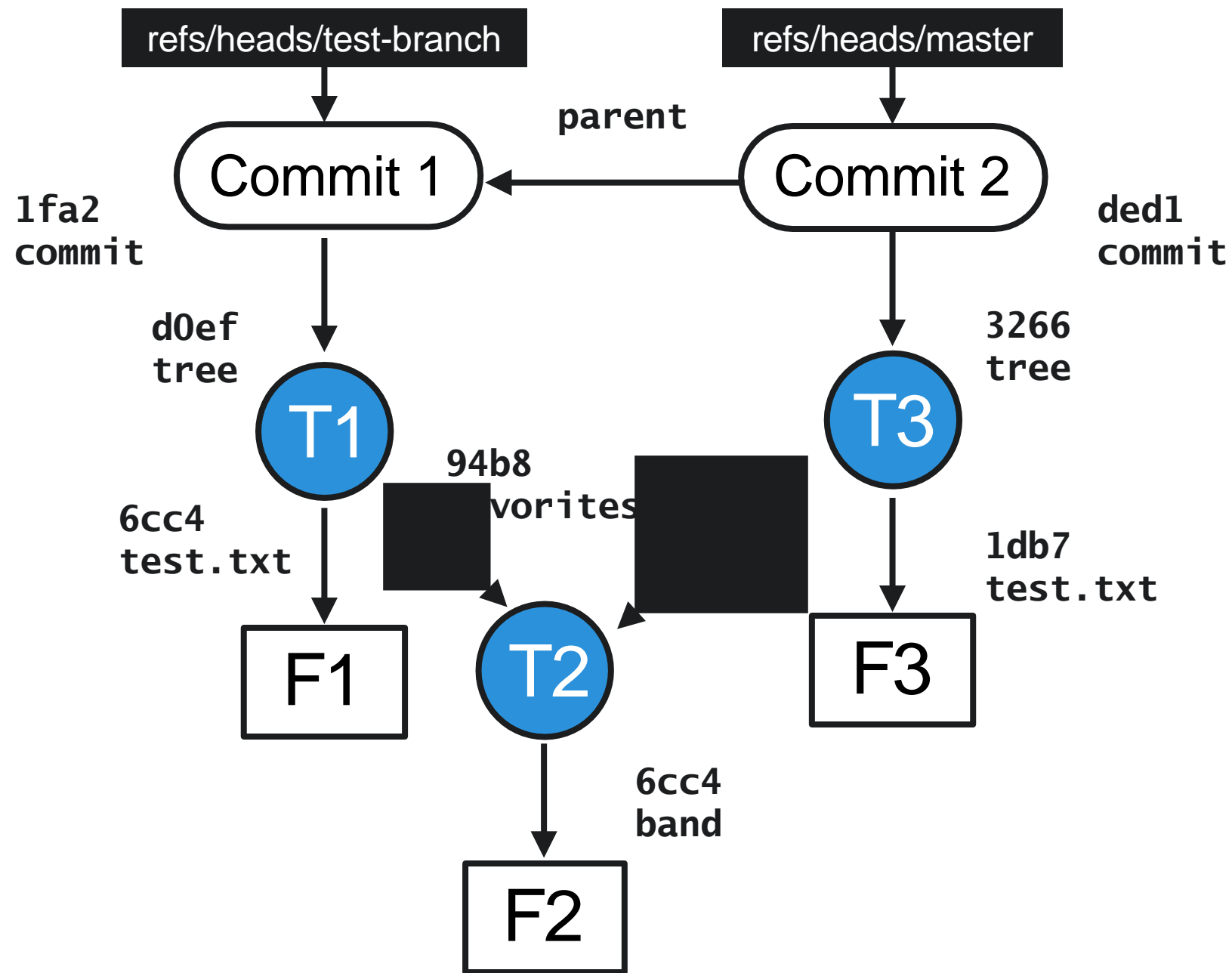
```
$ cat .git/HEAD  
ref: refs/heads/master
```

```
$ git branch  
test-branch  
* master
```

```
$ git symbolic-ref HEAD refs/heads/test-branch
```

```
$ cat .git/HEAD  
ref: refs/heads/test-branch
```

Data Structure



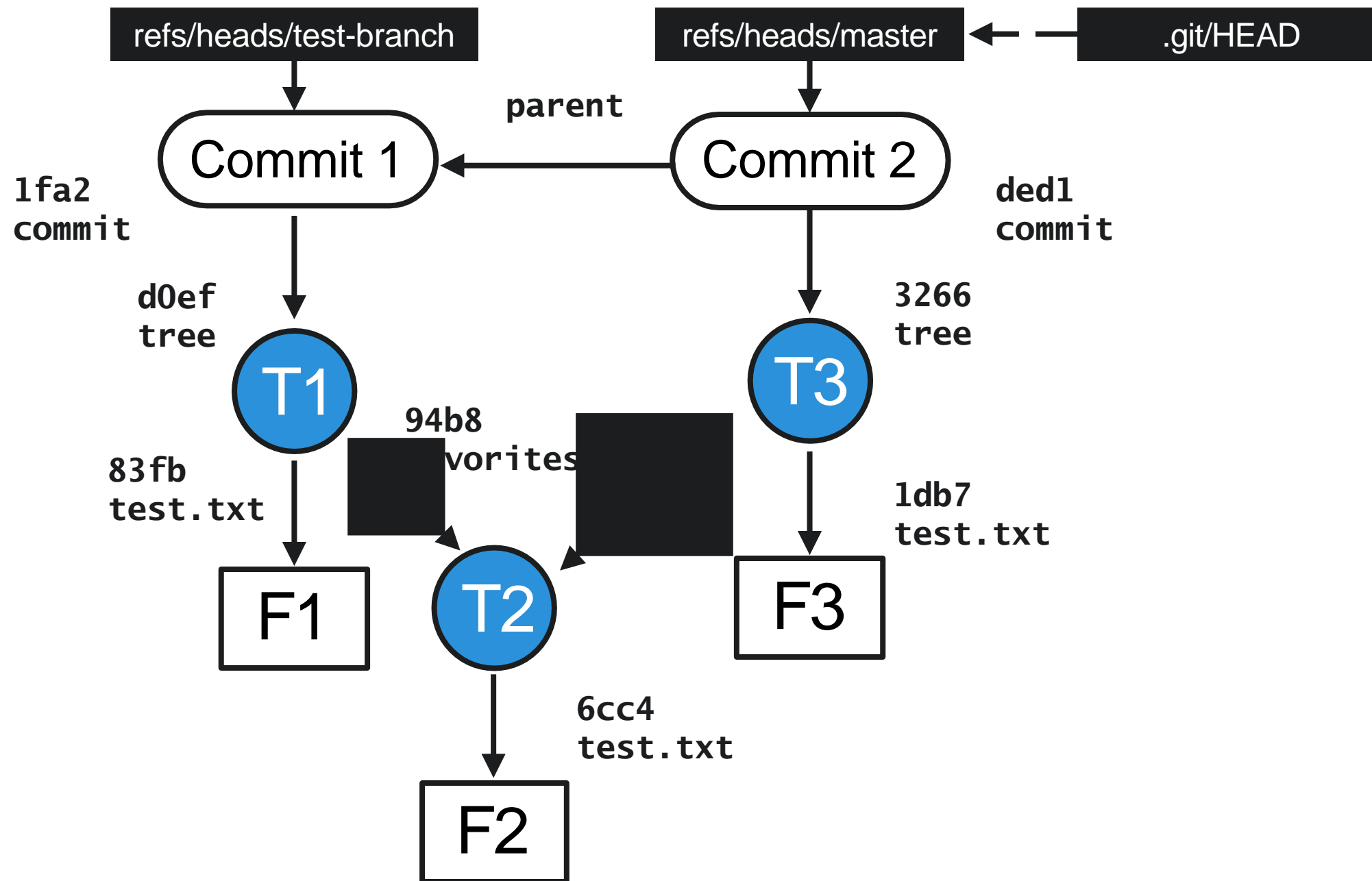
HEAD/head in Git

- HEAD is symbolic references. It points to another reference
- use ref, not SHA-1

```
$ cat .git/HEAD  
ref: refs/heads/master
```

```
$ git branch  
* master
```

Data Structure



Other **references** in Git

tag

once created, it will never change
unless explicitly updated with **--force**
useful for marking specific versions

Git Porcelain Commands

commit/push/pull/merge/...

- build upon plumbing commands

diff

\$ git diff HEAD

equivalent of...

\$ git ls-files -m

cherry

```
$ git cherry master origin/master
```

```
# equivalent of...
```

```
$ git rev-parse master..origin/master
```


status

```
$ git status
```

```
# equivalent of...
```

```
$ git ls-files --exclude-per-directory=.gitignore  
--exclude-from=.git/info/exclude --others --modified  
-t
```

commit

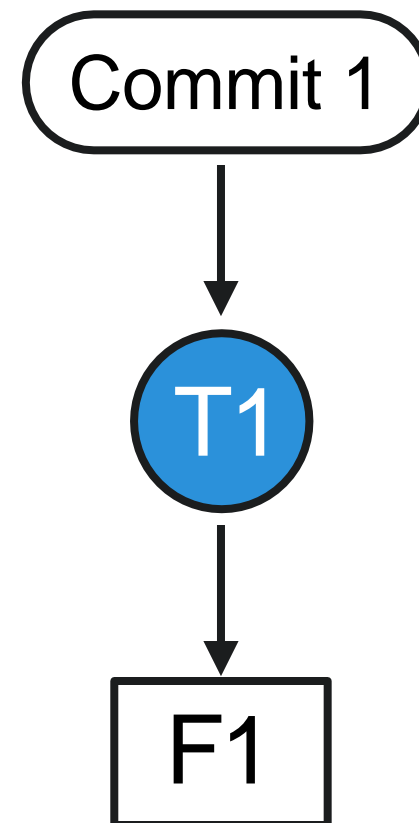
```
$ git update-index --add
```

F1

commit

```
$ git update-index --add
```

```
$ echo "commit-msg" |  
  git commit-tree  
[SHA-1]
```

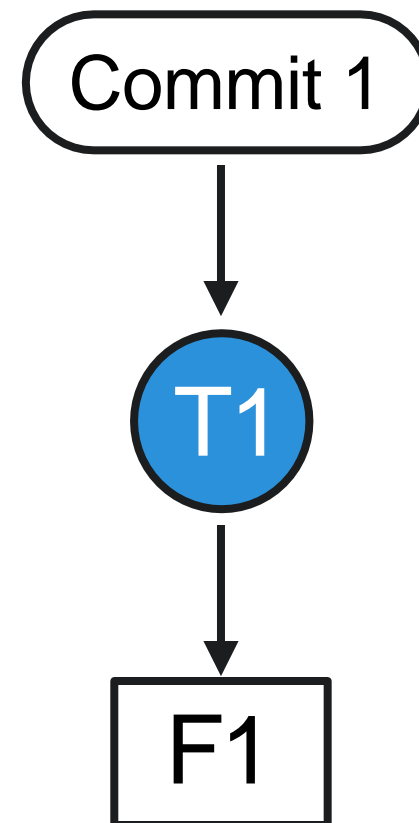


commit

```
$ git update-index --add
```

```
$ echo "commit-msg" |  
  git commit-tree  
[SHA-1]
```

```
$ git write-tree
```



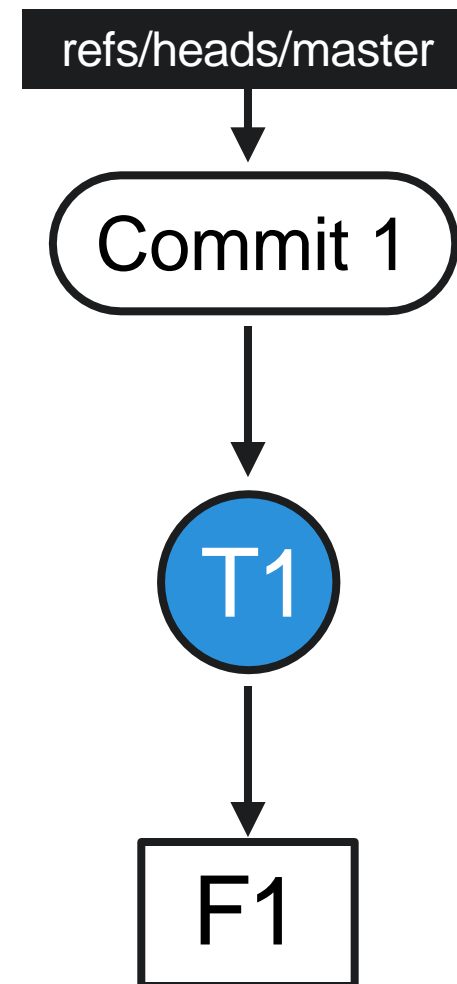
commit

```
$ git update-index --add
```

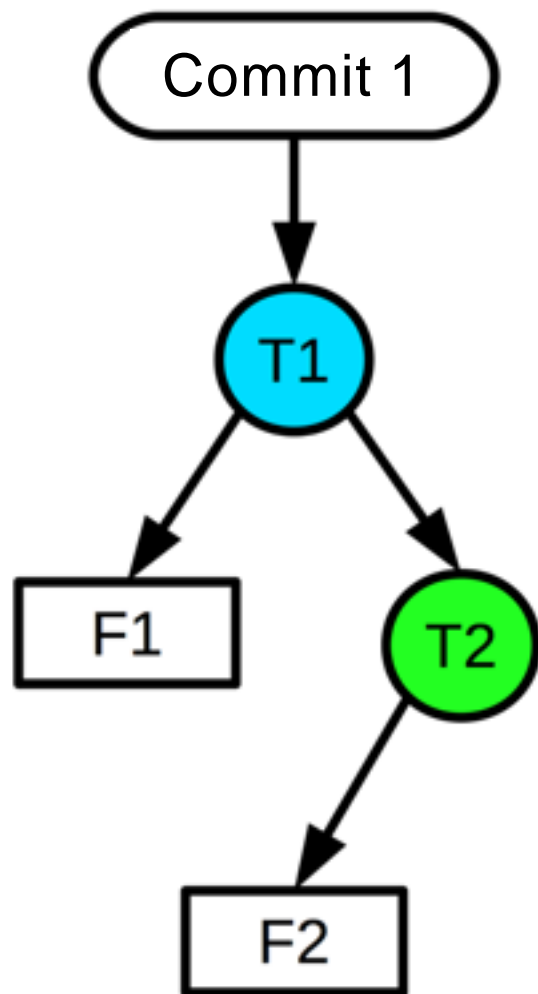
```
$ echo "commit-msg" |  
  git commit-tree  
[SHA-1]
```

```
$ git write-tree
```


```
$ git update-refs refs/  
heads/master
```



Commits in DAG

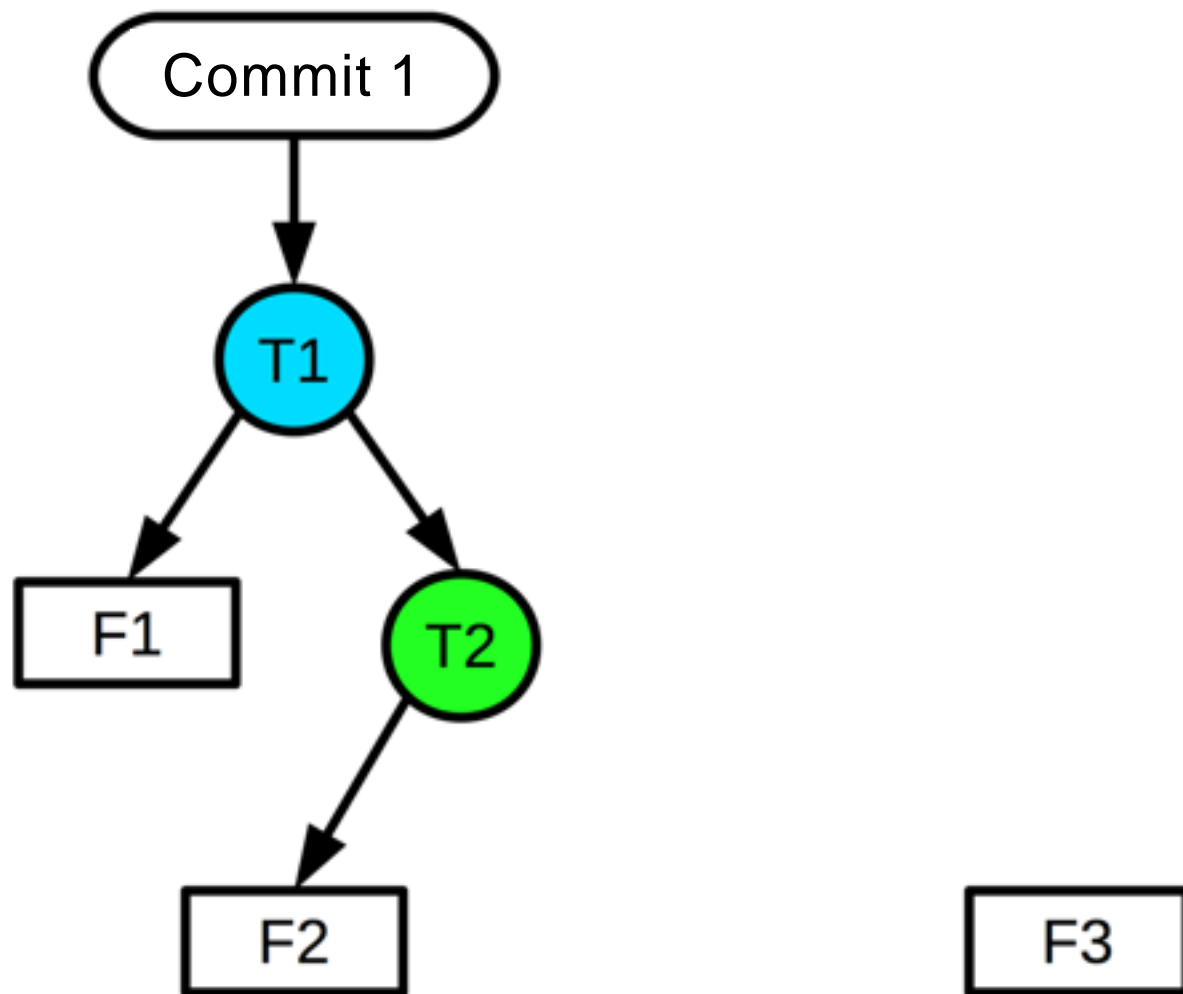


 : Commit

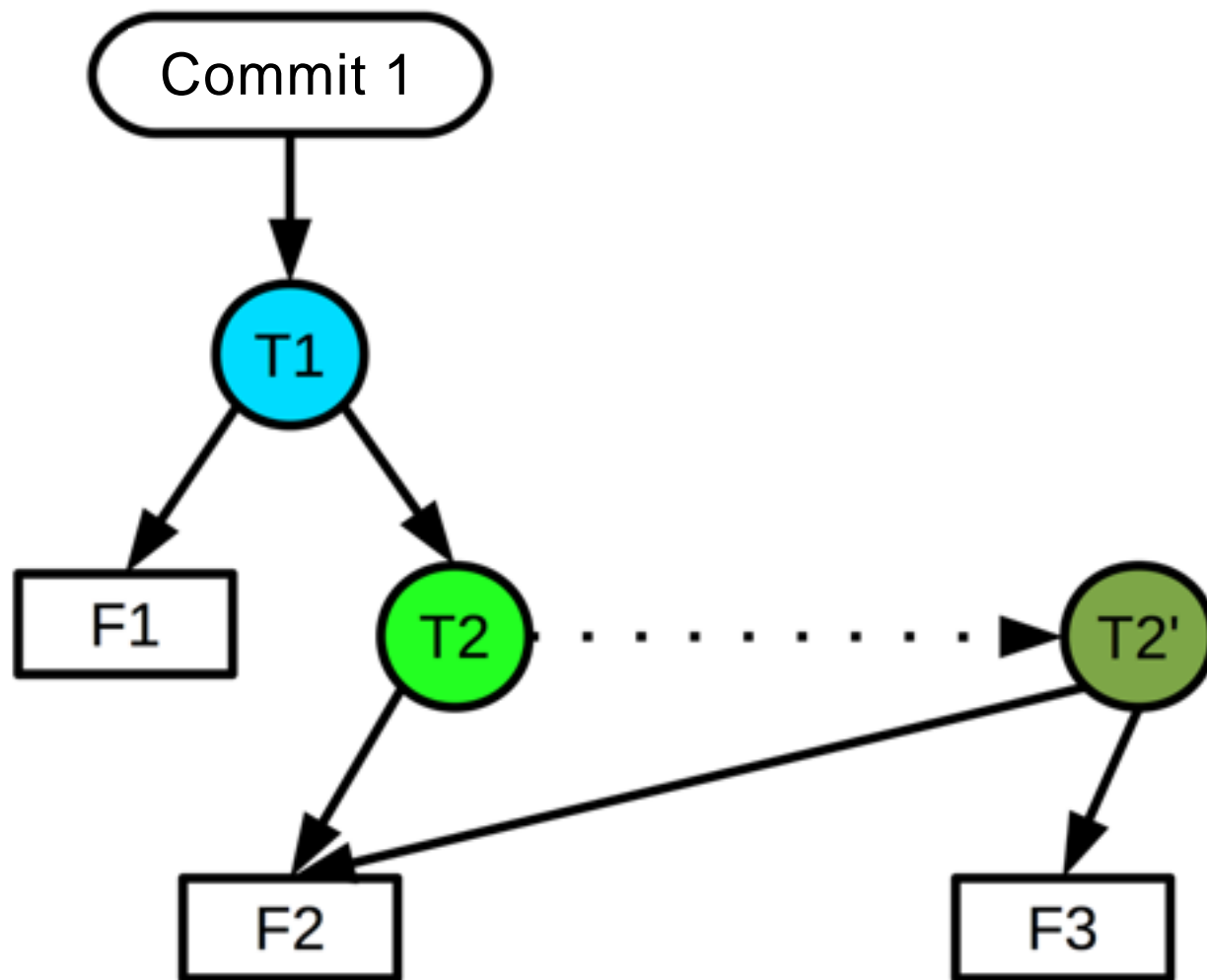
 : Tree info

 : Blob/File info

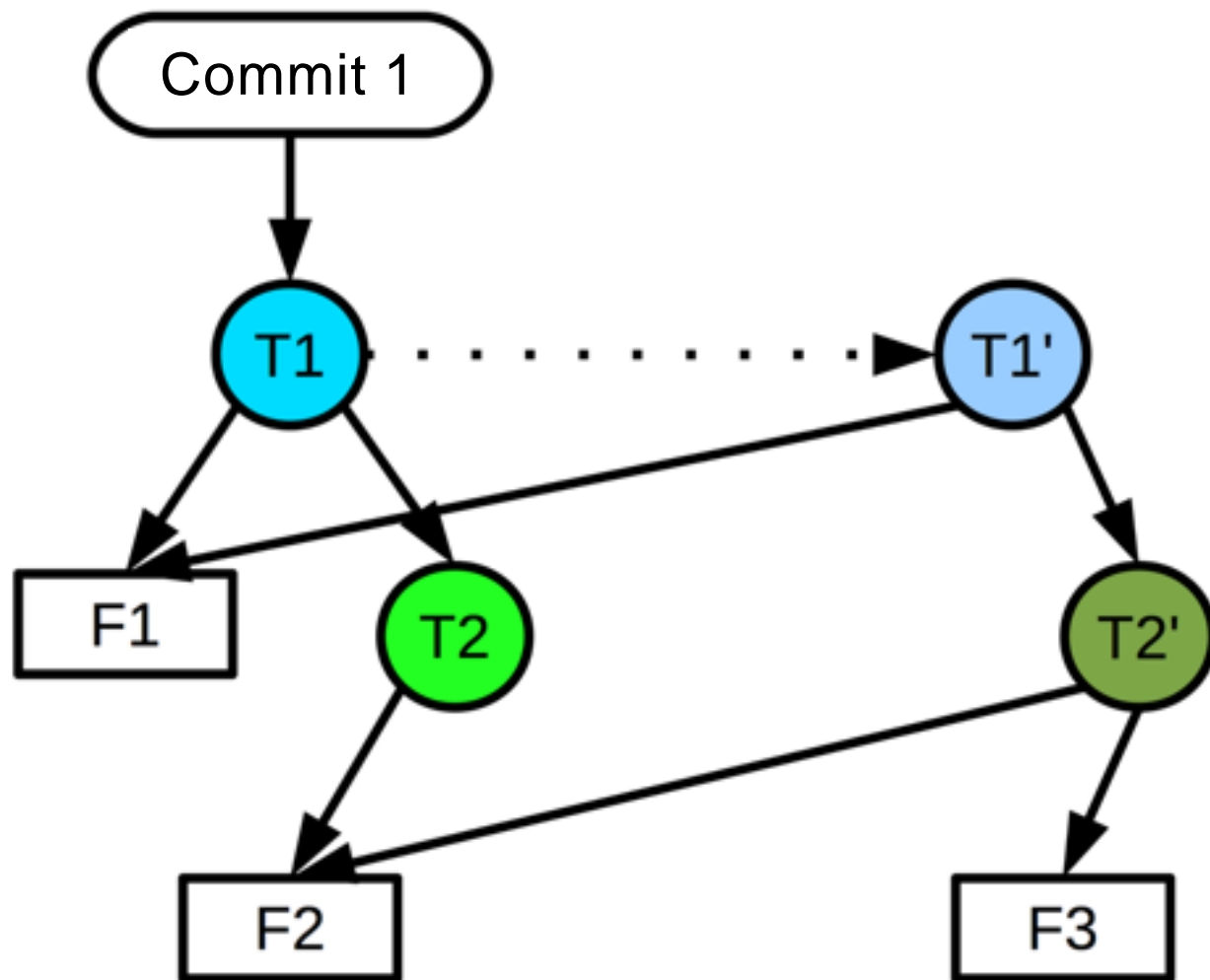
Create a New File (File 3)



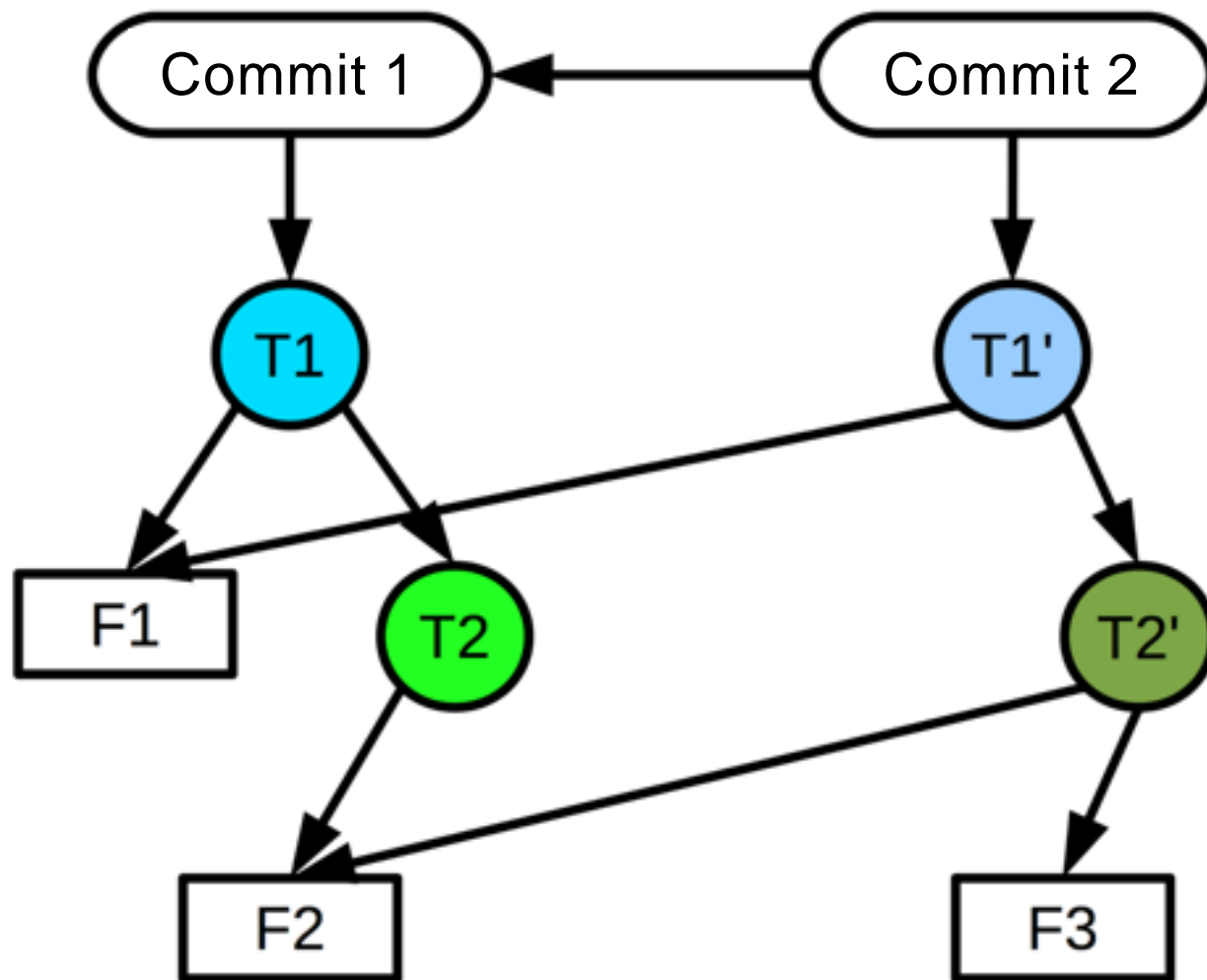
Create a New File (File 3)



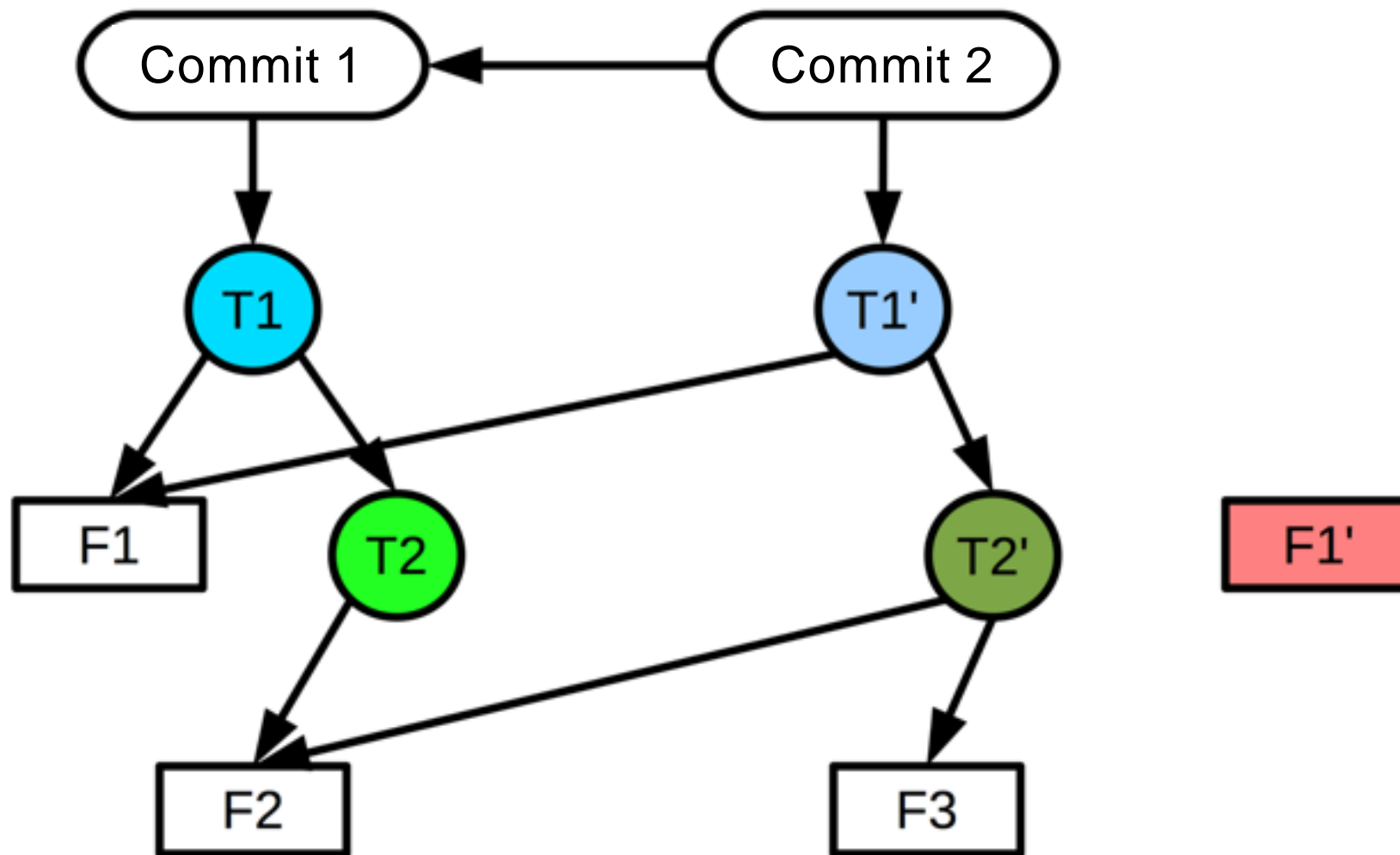
Create a New File (File 3)



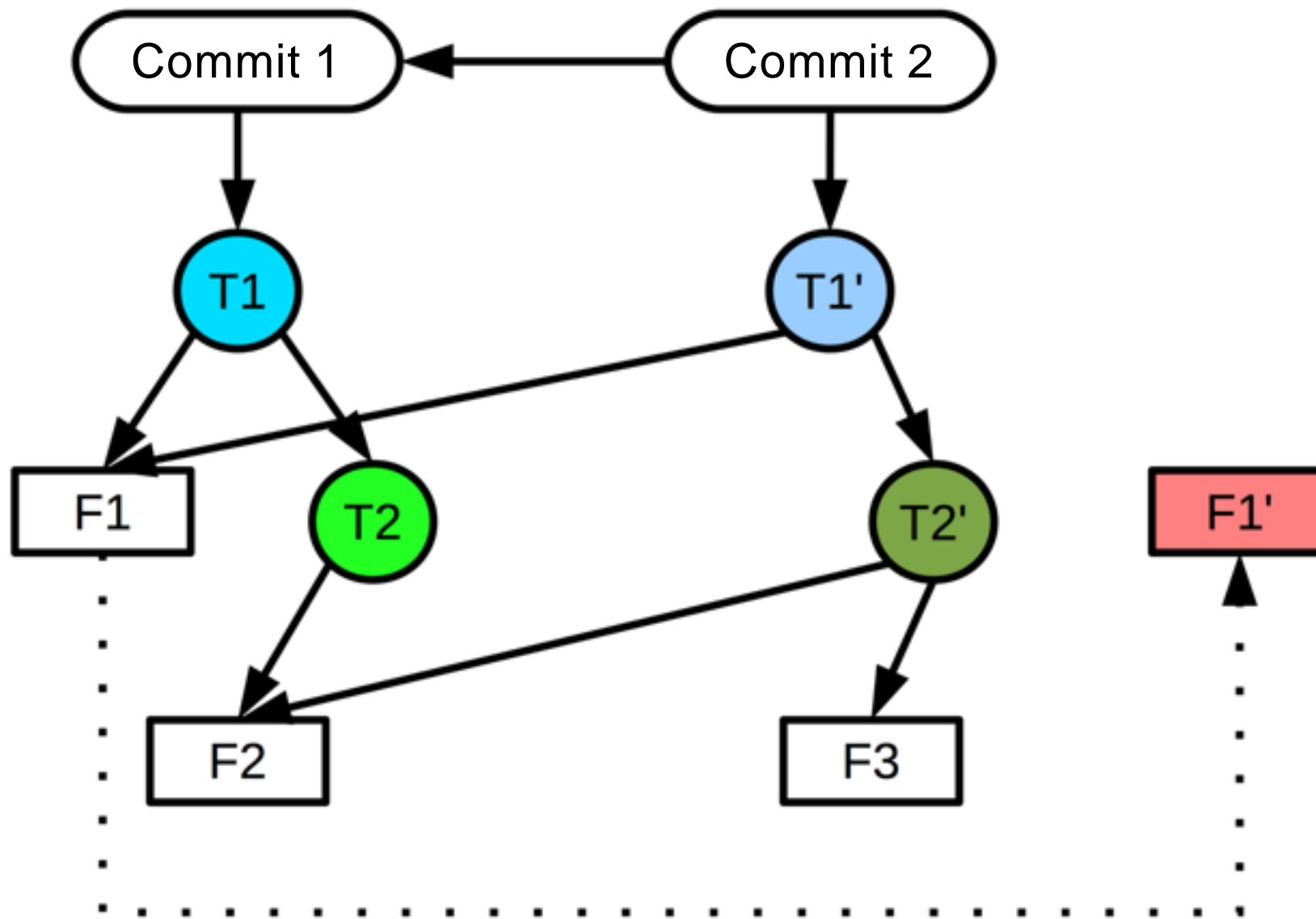
Commit a New File (File 3)



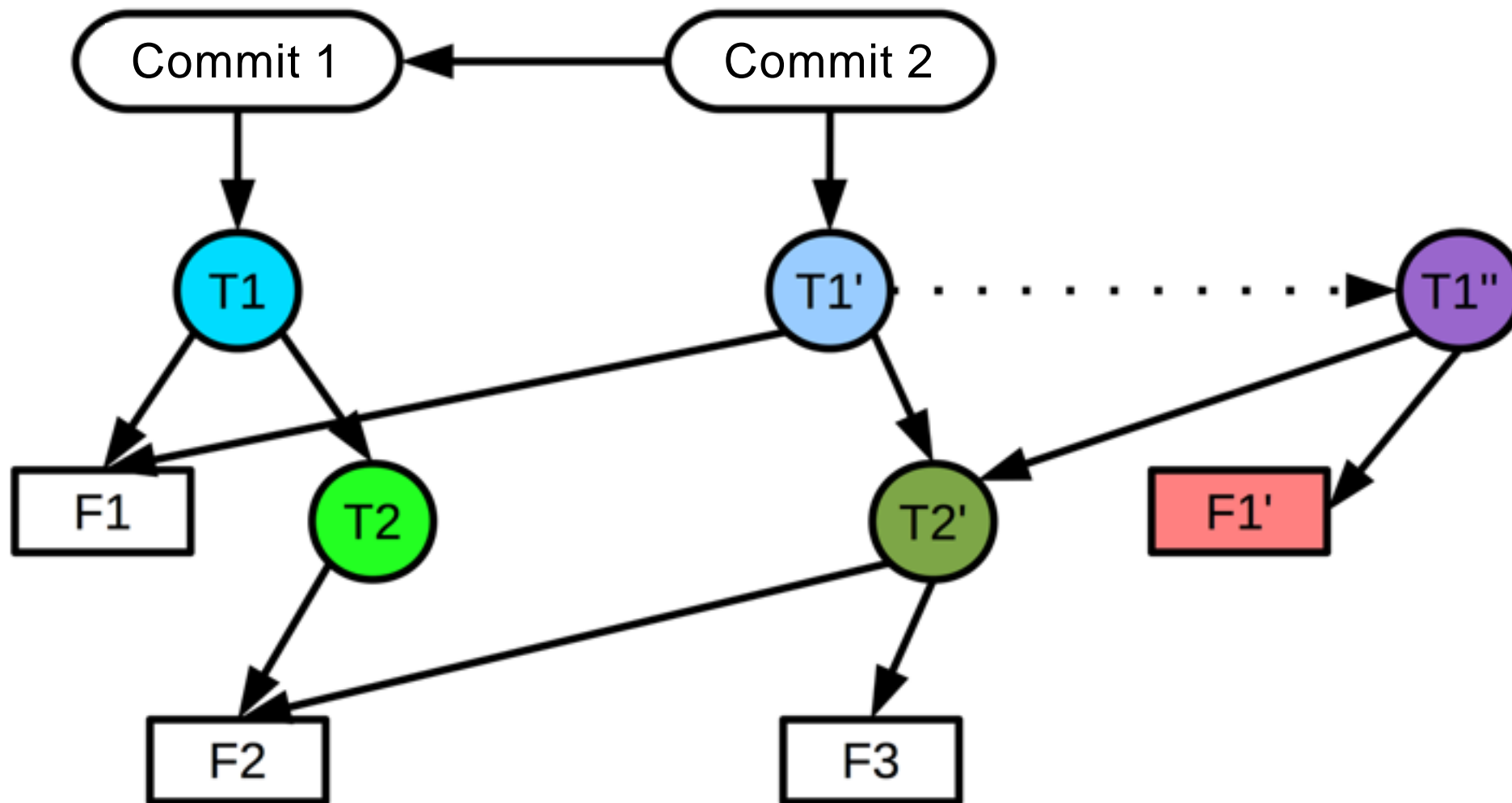
Edit and Commit File1



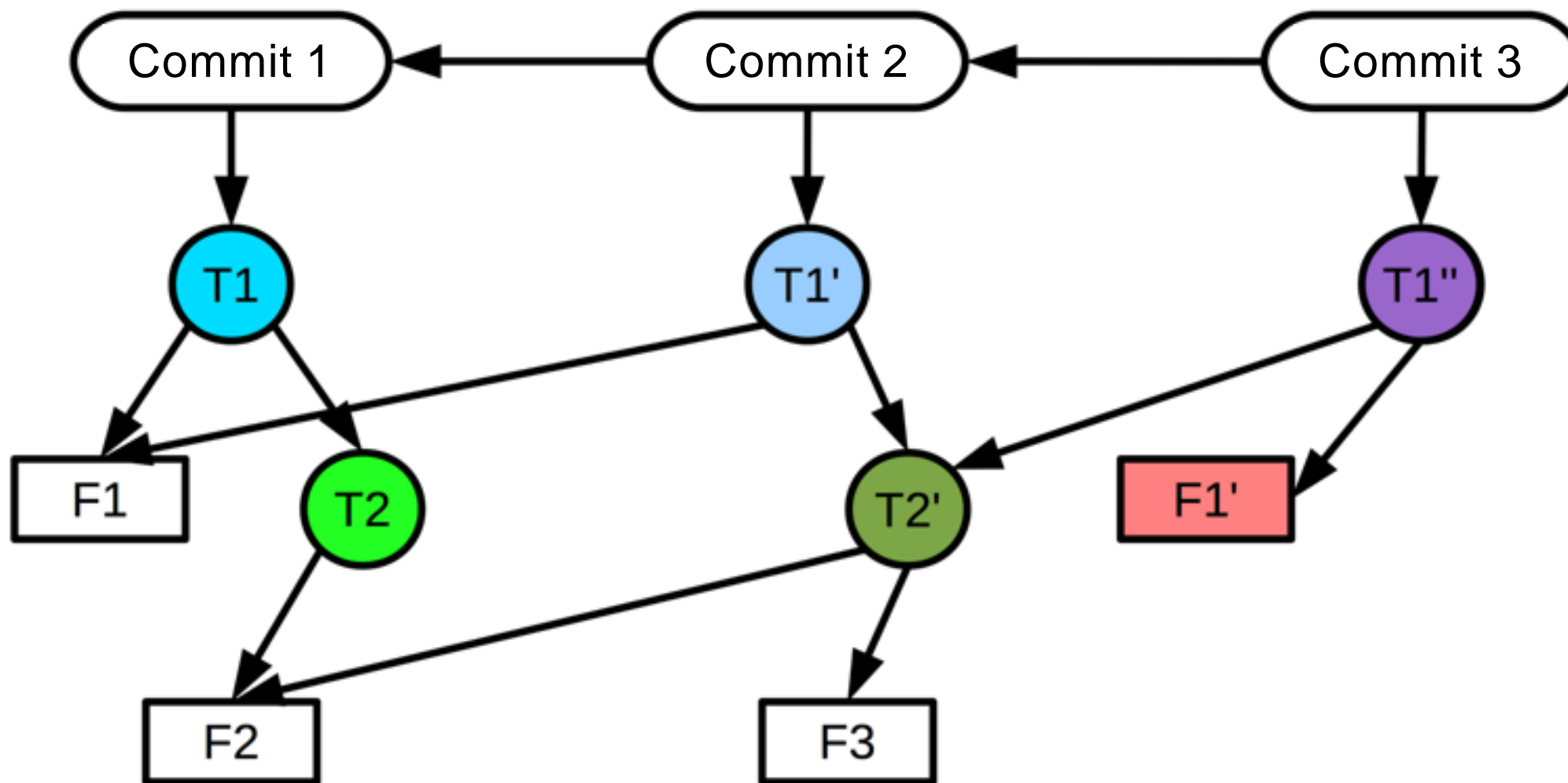
Edit and Commit File1



Edit and Commit File1

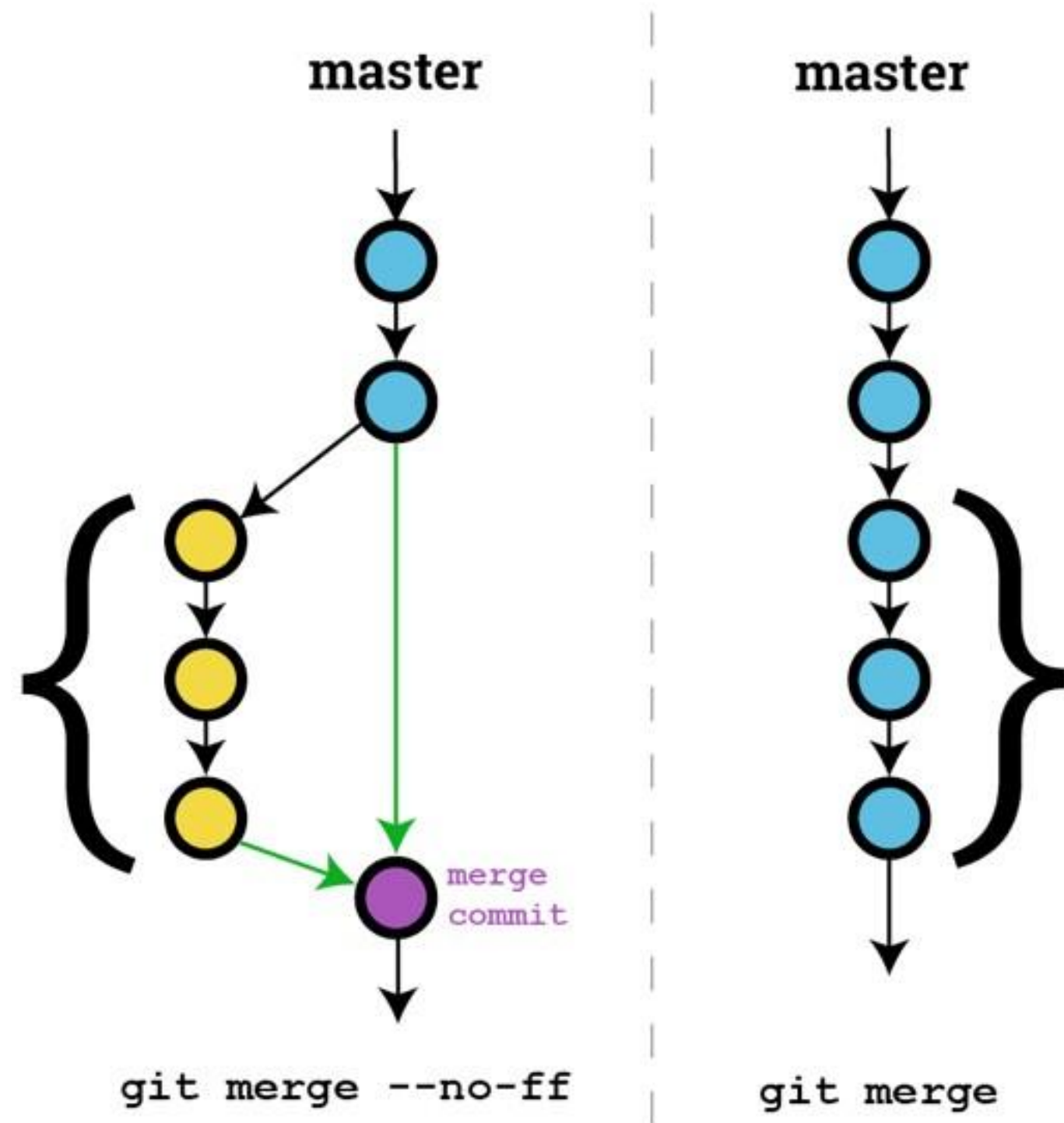


Edit and Commit File1



merge

merge is a commit with N parents,
retaining merge history



Basic Merge

```
$ git merge
```

```
$ git merge --squash
```


Basic Merge Conflicts

```
$ git status
```

```
On branch master
```

```
You have unmerged paths.
```

```
(fix conflicts and run "git commit")
```

```
Unmerged paths:
```

```
(use "git add <file>..." to mark resolution)
```

```
both modified:      index.html
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

Basic Merge Conflicts

```
<<<<<<< HEAD
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> some-remote-branch
```

```
<<<<<<< HEAD
(local) HEAD content is here
=====
some-remote-branch content is here
>>>>>>> some-remote-branch
```

Basic Merge Conflicts

```
$ git mergetool
```

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse
diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge

Merging:
index.html

Normal merge conflict for 'index.html':

{local}: modified file

{remote}: modified file

Hit return to start merge resolution tool (opendiff):

Aborting Merge

```
$ git merge --abort
```

Merge with Strategy

```
$ git merge -s ours
```

```
$ git merge -s theirs
```

pull

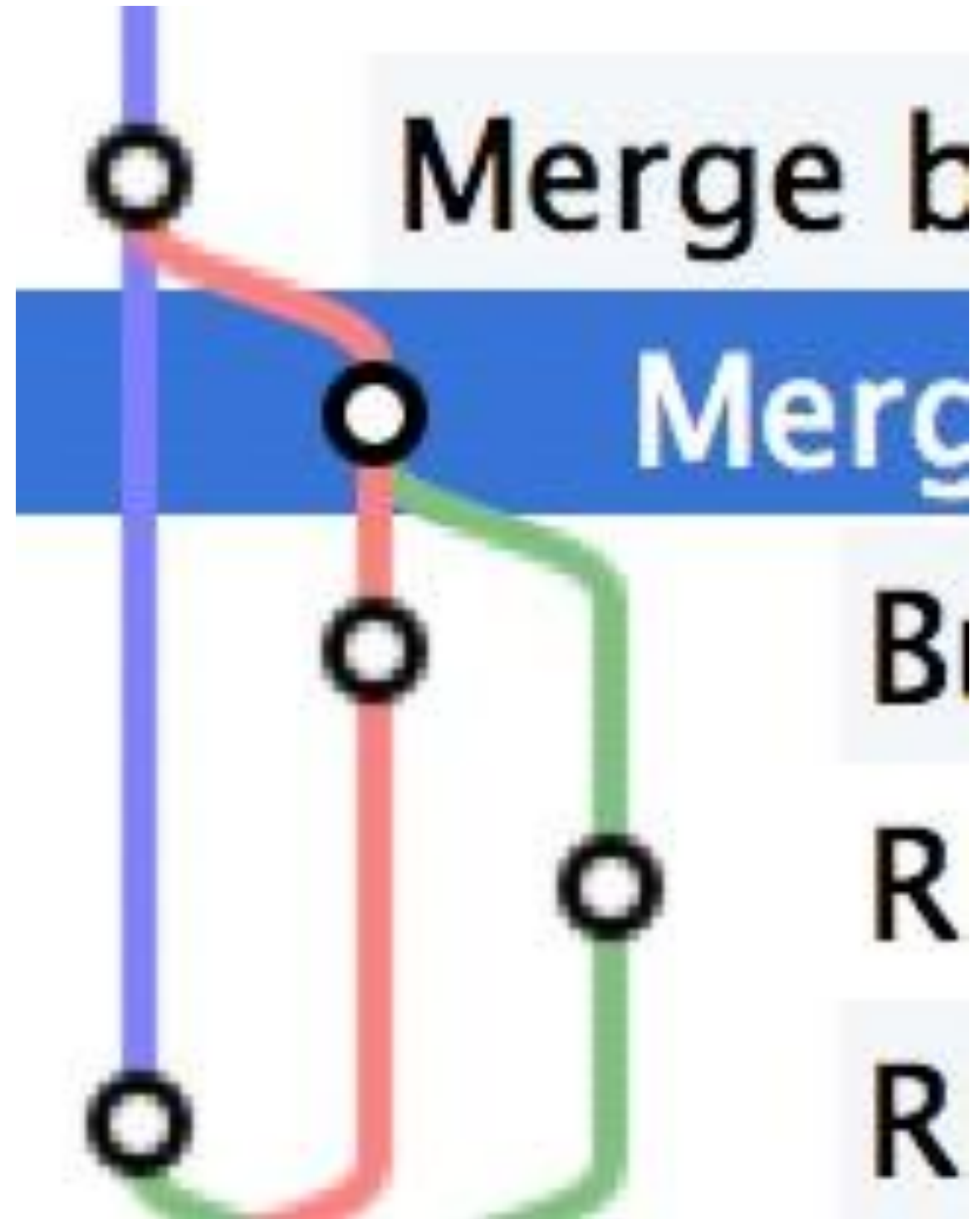
- fetch + merge
- conflict is possible

```
$ git pull origin/master
```

```
# equivalent of ...
```

```
$ git fetch origin/master
```

```
$ git merge origin/master
```



pull

- fetch + merge
- conflict is possible

\$ git pull

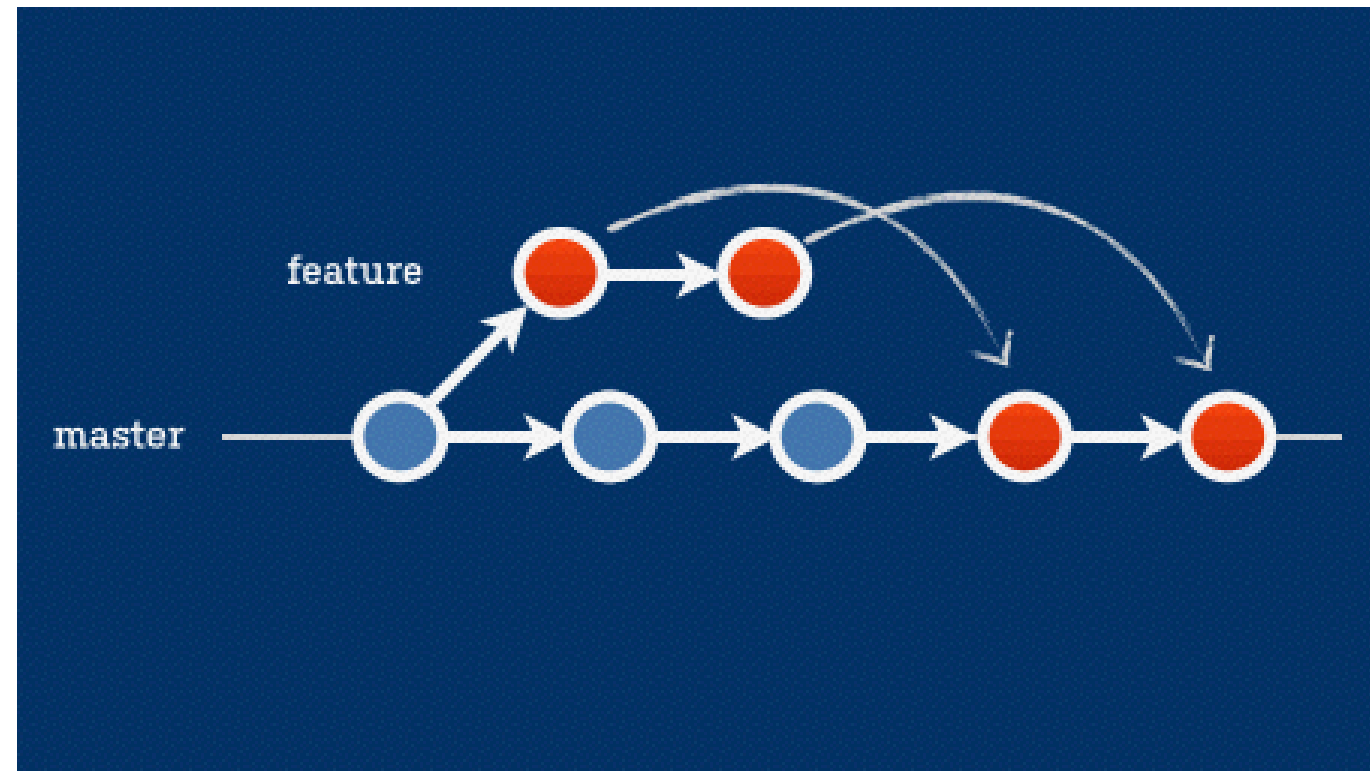
fetch

```
$ cat .git/config | grep remote
[remote "origin"]
  url = git://127.0.0.1/git/demo.git
  fetch = +refs/heads/*:refs/remotes/origin/*
```

source destination

rebase

replay commits one by one
on top of a branch



Changing History is...

- fun
- dangerous
- use with caution, esp. working with others
- only use in local repositories/branches

rebase

equivalent to `git pull --rebase`

\$ `git fetch origin/master; git rebase origin/master`

A---B---C topic
/
D---E---F---G master

to:

A'--B'--C' topic
/
D---E---F---G master

rebase

```
$ git rebase -i
```

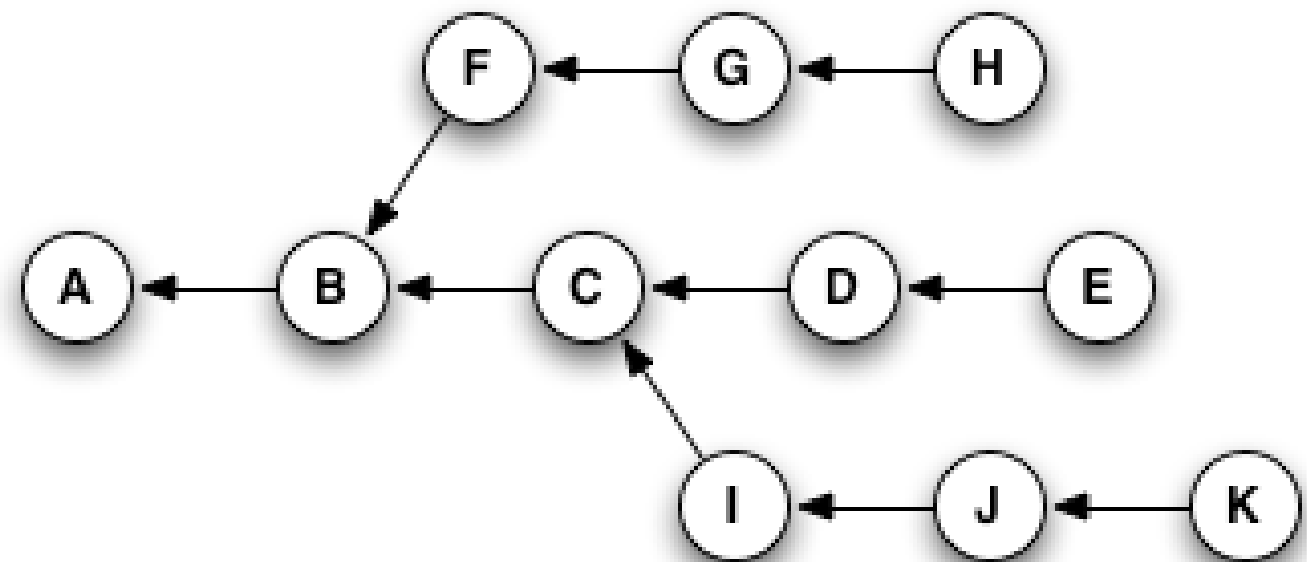
```
$ git rebase --onto
```

rebase

- replays commits
- moves commits
- changes commit history
- changes commit parent/ancestor

Branches in Git

- just like parallel universes
- nothing more than references to commit



List all Branches

list local branches

\$ git branch

**# list local and remote branches
t**

\$ git branch -a

show branches with reference

\$ git branch -v

show branch tracking info

\$ git branch -vv

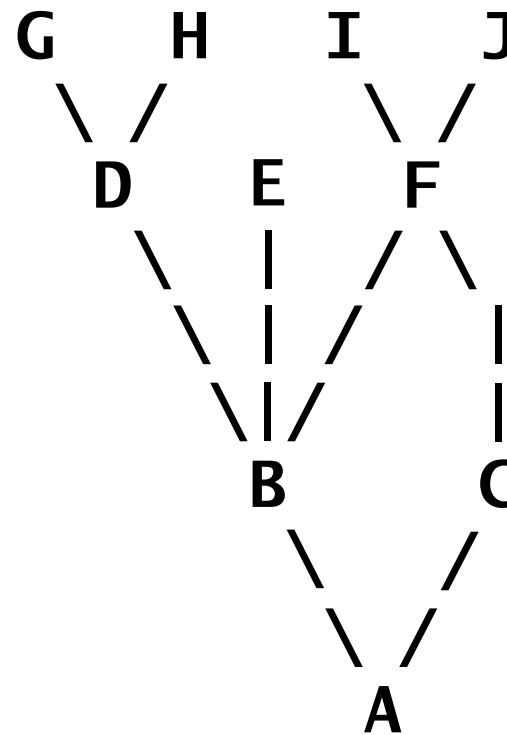
Setting Tracking Info

```
$ git remote -vv
```

```
$ git remote set-url http://example.com/foo.git
```

```
$ git remote add staging git://git.kernel.org/.../gregkh/staging.git
```

```
# or just edit .git/info
```

Caret and Tilde

- **caret(~n)**: depth-first *n*-th parent commit
- **tilde(^n)**: breadth-first *n*-th parent commit
- See `$ git help rev-parse`
Revision Selection

A	=	=	A⁰	
B	=	A¹	=	A~1
C	=	A²	=	
D	=	A¹¹	=	A~2
E	=	A¹²	=	
F	=	A¹³	=	
G	=	A¹¹¹	=	A~3
H	=	B¹²	=	A¹¹² = A~2²
I	=	B¹³	=	A¹¹³
J	=	B¹³²	=	A¹¹³²

push . default

nothing

do not push anything (error out) unless a refspec is explicitly given. This is primarily meant for people who want to avoid mistakes by always being explicit.

current

push the current branch to update a branch with **the same name on the receiving end**. Works in both central and non-central workflows.

upstream

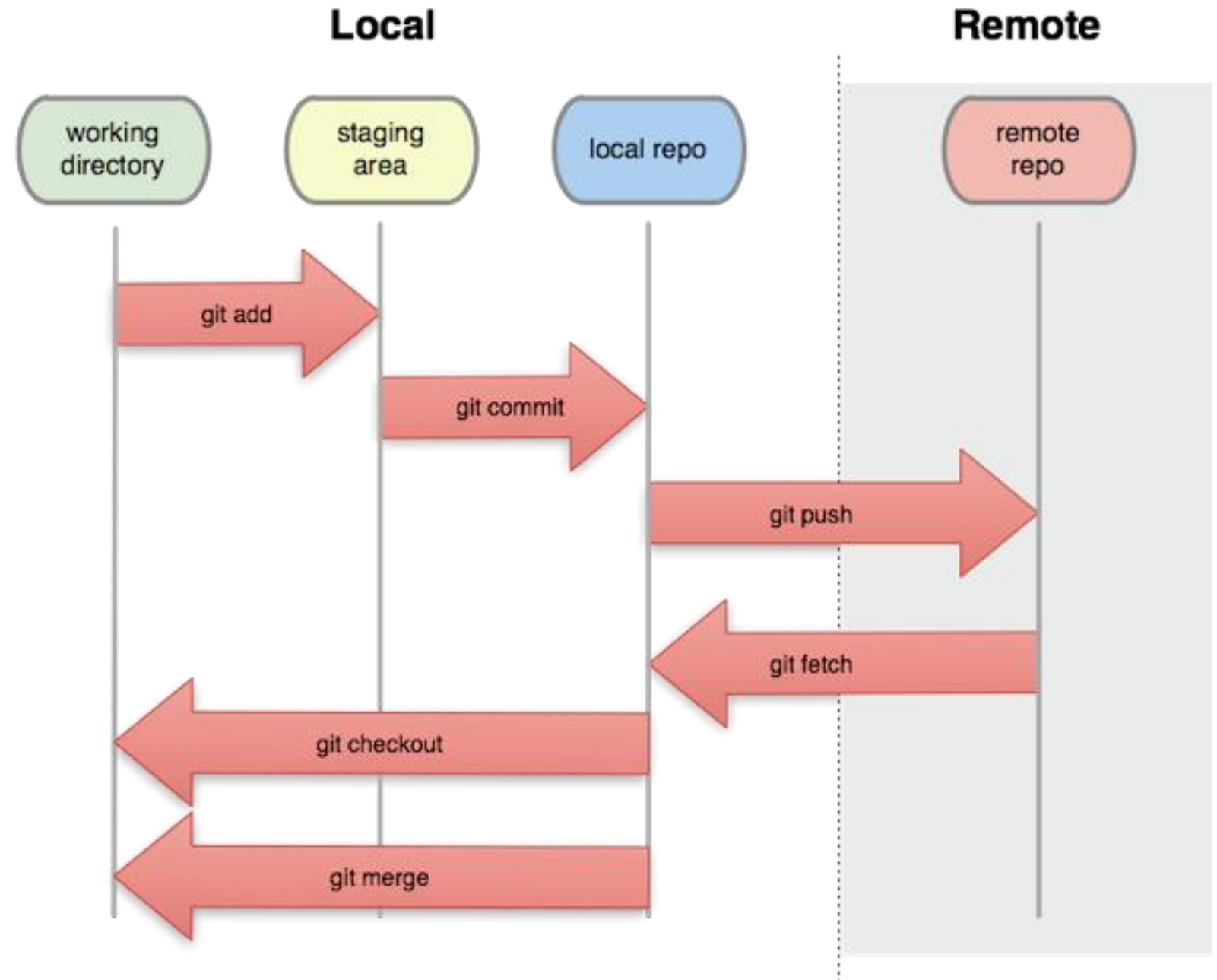
push the current branch back to the branch whose changes are usually integrated into the current branch (which is called `@{upstream}`). This mode only makes sense if you are pushing to the same repository you would normally pull from (i.e. central workflow).

simple

in centralized workflow, work like upstream with an added safety to refuse to push if the upstream branch's name is different from the local one. When pushing to a remote that is different from the remote you normally pull from, work as current.

This is the safest option and is suited for beginners.

Cheat Sheet



stash

working
area

staging
area

local
repository

upstream
repository

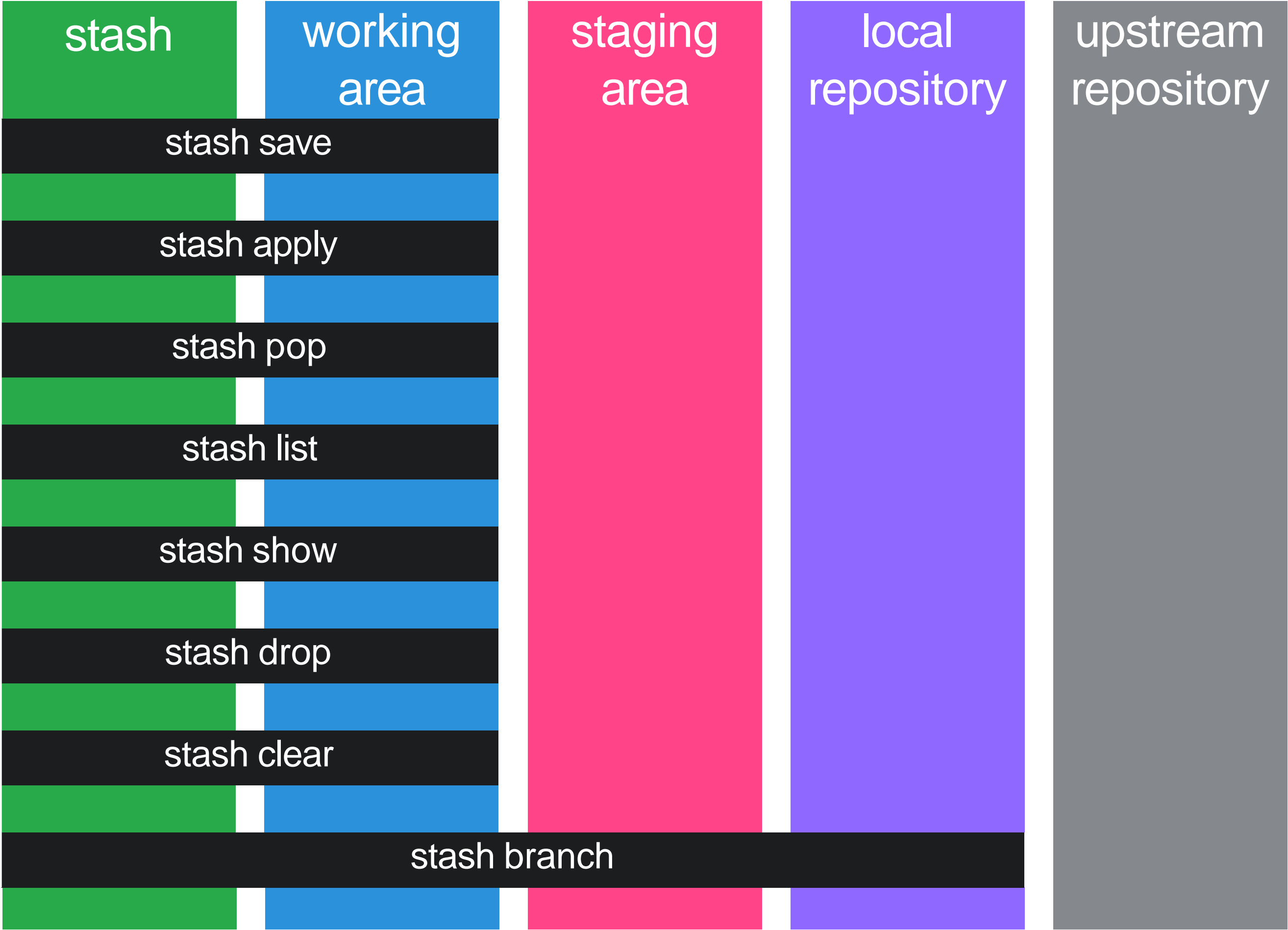
stash

working
area

staging
area

local
repository

upstream
repository



stash

working
area

staging
area

local
repository

upstream
repository



stash

working
area

staging
area

local
repository

upstream
repository

stash

working
area

staging
area

local
repository

upstream
repository

reset HEAD

reset —soft

diff —cached

commit

stash

working
area

staging
area

local
repository

upstream
repository

stash

working
area

staging
area

local
repository

upstream
repository

log

branch

fetch

push

push — delete

stash

working
area

staging
area

local
repository

upstream
repository

stash

working
area

staging
area

local
repository

upstream
repository

branch -r

Find Stuff in Git

Find Stuff by...

find file by content

\$ git grep

find file by its name

\$ git ls-file | grep

find changes by line number

\$ git blame -L

Find Stuff in **log**

find changes by commit message

\$ git log | grep

find changes with specific string

\$ git log -S

find changes by file name

\$ git log -p

find changes by author/commmitter

\$ git log --author

\$ git log --commmitter

#find changes within given time span

\$ git --since

\$ git --until

More about **log**

ignore merges

\$ git log --no-merges

customize how log looks

\$ git log --graph

\$ git log --oneline

\$ git log --pretty="..."

\$ git log --stat

Undo & Cleanup in Git

Change Last Commit

```
$ git commit --amend
```

```
$ git commit --amend --no-edit
```

```
$ git commit ; git rebase -i HEAD~2
```

Unstage Staged Files

```
$ git reset HEAD [filename]
```

```
$ git reset .
```

Undo Working Modifications

```
$ git checkout -- [filename]
```

```
$ git checkout .
```

```
$ git stash
```

```
$ git stash drop
```

Cleanup Untracked Files

```
$ git clean -df
```

```
$ git add .
```

```
$ git stash
```

```
$ git stash drop
```

Delete Remote Branch

```
# check for proper remote branch name
```

```
$ git branch -a
```

```
$ git push origin --delete [branch_name]
```

```
$ git push origin :[branch_name]
```

```
# ----- Explanation ----- #
```

```
$ cat .git/config | grep remote
```

```
[remote "origin"]
```

```
    url = git://127.0.0.1/git/demo.git
```

```
    fetch = +refs/heads/*:refs/remotes/origin/*
```

source

destination

```
# pushing empty source/reference to destination
```


Be the Last Committer in a Branch

```
# use with caution  
$ git commit --allow-empty
```

Basic Recoveries in Git

Recover Deleted Branch

```
$ git branch -d test-branch  
Deleted branch test-branch (was 1fa215b).
```

```
$ git checkout 1fa215b  
HEAD is now at 1fa215b... test commit
```

```
$ git co -b test-branch
```

- Do it before garbage collection

Reset Indices

\$ git reset --soft

Does not touch the index file or the working tree at all (but resets the head to <commit>, just like all modes do). This leaves all your changed files "Changes to be committed", as git status would put it.

\$ git reset --mixed

Resets the index but not the working tree (i.e., the changed files are preserved but not marked for commit) and reports what has not been updated.

This is the default action.

\$ git reset --hard

Resets the index and working tree. Any changes to tracked files in the working tree since <commit> are discarded.

Diagnose Problems

```
$ git reflog
```

```
1fa215b HEAD@{0}: checkout: moving from 1fa215b to test-branch
```

```
1fa215b HEAD@{1}: checkout: moving from master to 1fa215b
```

```
ded15c0 HEAD@{2}: 
```

```
...
```

Extending Git

Put `git-[command]` in your `$PATH`

```
# show differences between two branches, group by commit  
$ git peach
```

```
ded15c0  another test commit (2 days ago) <Jingwei "John" Liu>  
test.txt | 2 +-  
1 file changed, 1 insertion(+), 1 deletion(-)
```

Put `git-[command]` in your `$PATH` (cont)

```
#!/usr/bin/env bash
# Author: Jingwei Liu <liujingwei02@meituan.com>
# Last Modified: Mar 26 2014 05:52:42 PM
set -e

# show changed files in git cherry
if [[ -n $1 ]]; then
    if [[ $1 =~ ^[0-9]*$ ]] ; then
        upstream="HEAD~$1"
    else
        upstream=$1
    fi
else
    upstream="master"
fi
git cherry $upstream | awk '{print $2}' | xargs git show --stat --
pretty=format: '%Cred%h%Creset %Creset %s %Cgreen(%cr) %C(bold blue)<
%an>%Creset'
```


Other...

- Hooks
- Aliases
- Config
- Submodule

References

- git-scm.com
- gitready.com
- ProGit
- [Deep Darkside of Git](#) (source of a large portion of slides)
- Just use it, and read **git help [command]**

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Thank you!