Revision control
Advanced **git**

Waterford Institute of Technology

April 30, 2016

John Fitzgerald

Presentation outline

Estimated duration presentation

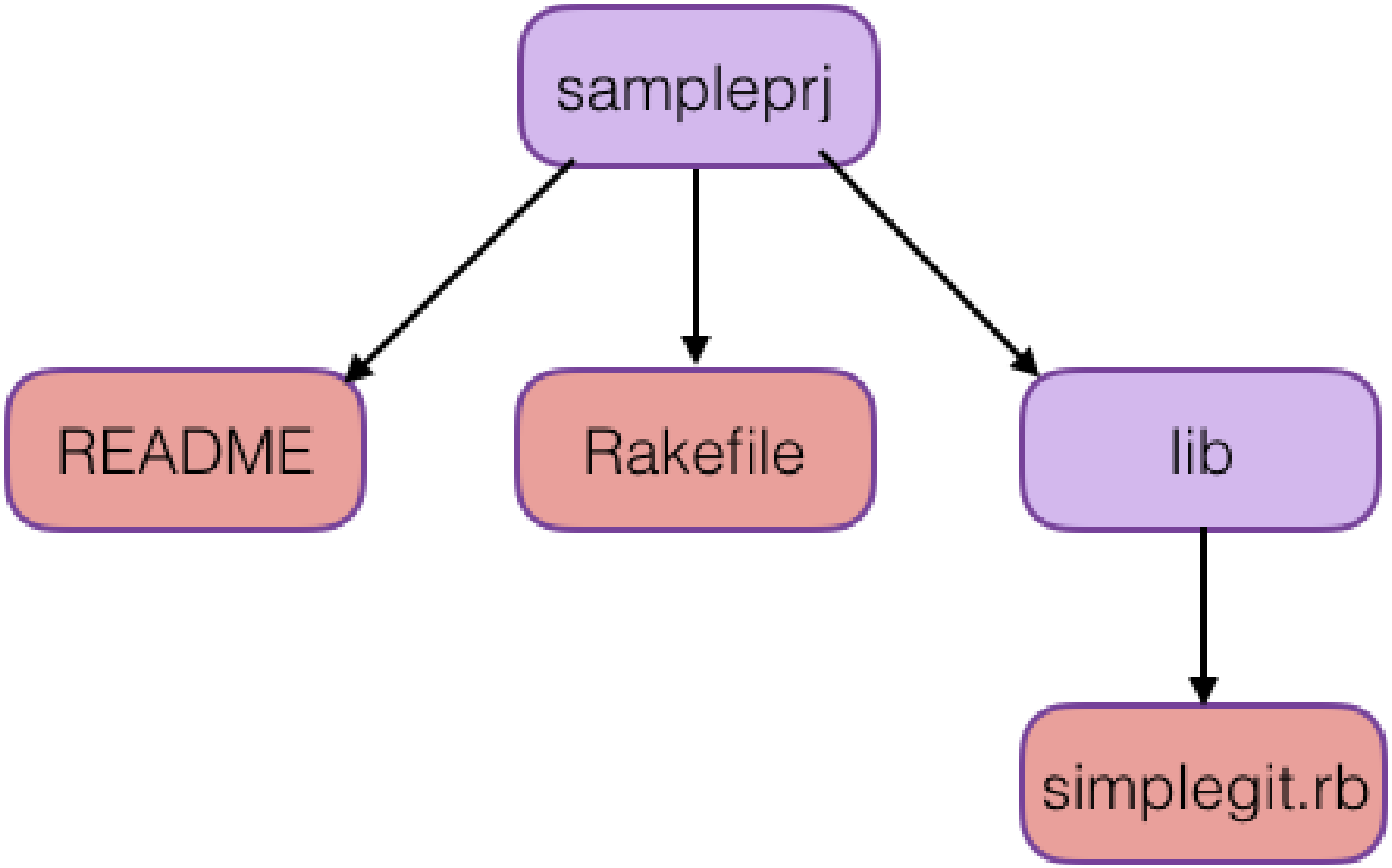
Questions at end presentation

Topics discussed:

- Git internals brief exploration
- Merging
- Rebasing
- Resolving conflicts
- Squash commits

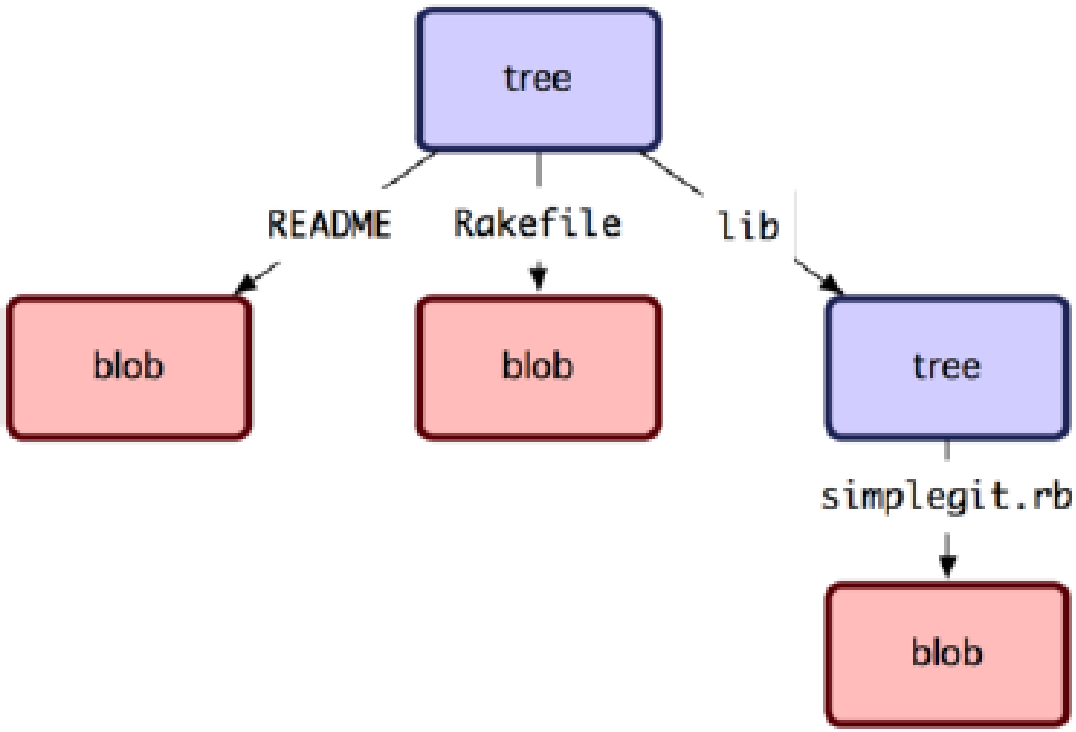
Sample project

File structure



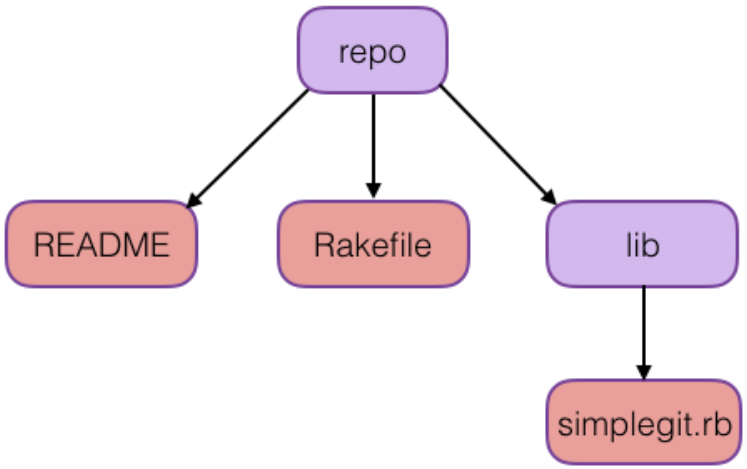
Sample project

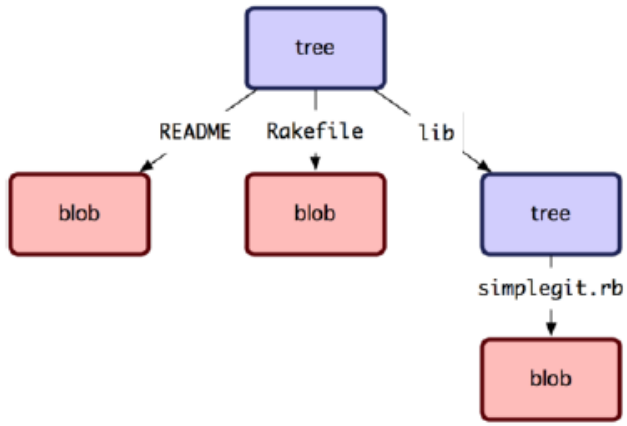
Git structure



Sample project

Files, folders & Git structure





Git objects

Tree, Blob, Commit, Tag

Git has 4 objects:

- **Tree**: equivalent to file or folder
- **Blob**: stores data
- **Commit**: describes the commit
- **Tag** (light & annotated): describes the tag

Git objects

How content is stored

Git stores content in Tree & Blob objects

- **Tree**
  - Faclitates storage group files
  - Similar to folder or directory
- **Blob**
  - Corresponds to file contents
  - May include trees (subfolders)

Git objects

Commit

```
git log

commit 2fc01ef0ccd93776fe151d1e47e09b08067e1d4c
Author: Your Name <yourEmail@witmail.com>
Date:   Sat Apr 23 19:36:51 2016 +0100

    your commit message
```

Git objects

Annotated tag

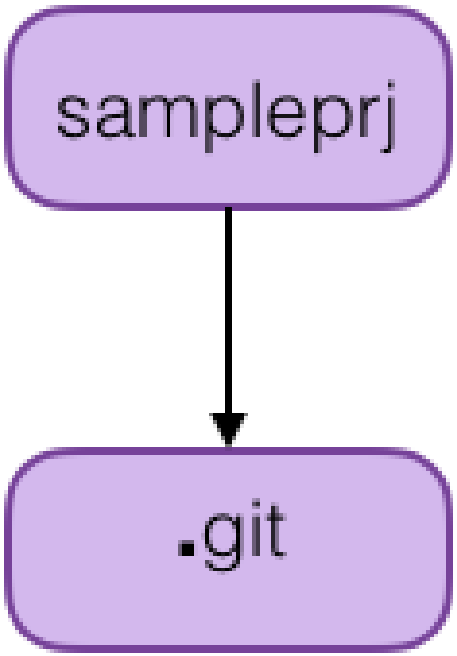
```
git tag -a v1
git show v1

tag v1
Tagger: Your Name <yourEmail@witmail.com>
Date:   Tue Apr 26 18:36:42 2016 +0100

your tag message

commit 7b15dfa7c41ee17c8ef27808c1043df8f91d4aa7
```

Sample project

Git init

Initialize empty .git repository

```
git init
```

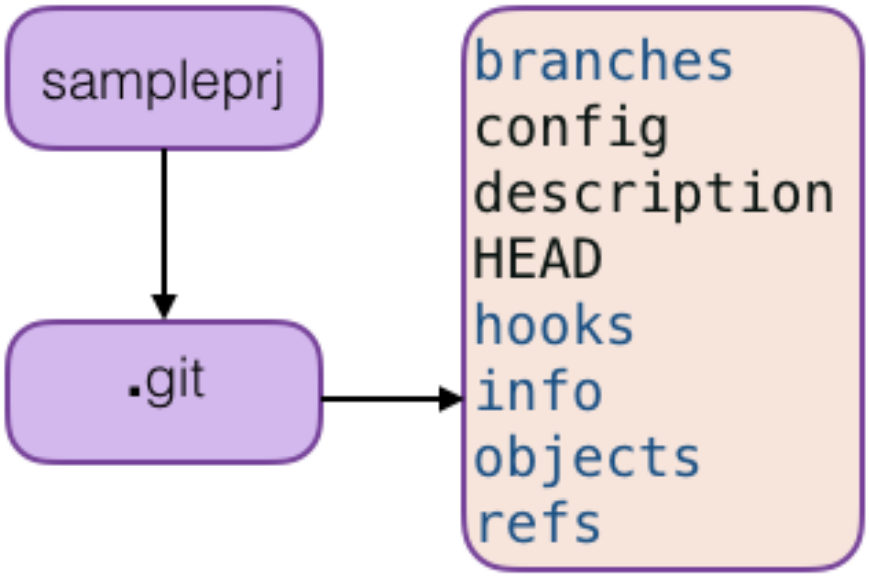


Sample project

Git init

Uninititlized **.git** content

```
ls -f1 .git
```



.git folder

Repository populated

Additional files and folders in **.git** when content committed

Core git components:

HEAD: Points to checked out branch
index: Staging area storage
refs: Commit object pointers
objects: Stores content

- tree, blob, commit, tag

HEAD

Points to checked out branch

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

```
git checkout -b dev
Switched to new branch dev
cat .git/HEAD
ref: refs/heads/dev
```

```
git branch

* dev
  master
```

index

README: not tracked, not staged file

```
echo Sample project > README
git status

On branch master
Untracked files:
  (use "git add <file>..." to include in
   what will be committed)

      README

nothing added to commit but untracked
files present (use "git add" to track)
```

index

README: a staged file

```
git add README
git status

On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

refs

Pointers to commit objects

Examples **refs** values:

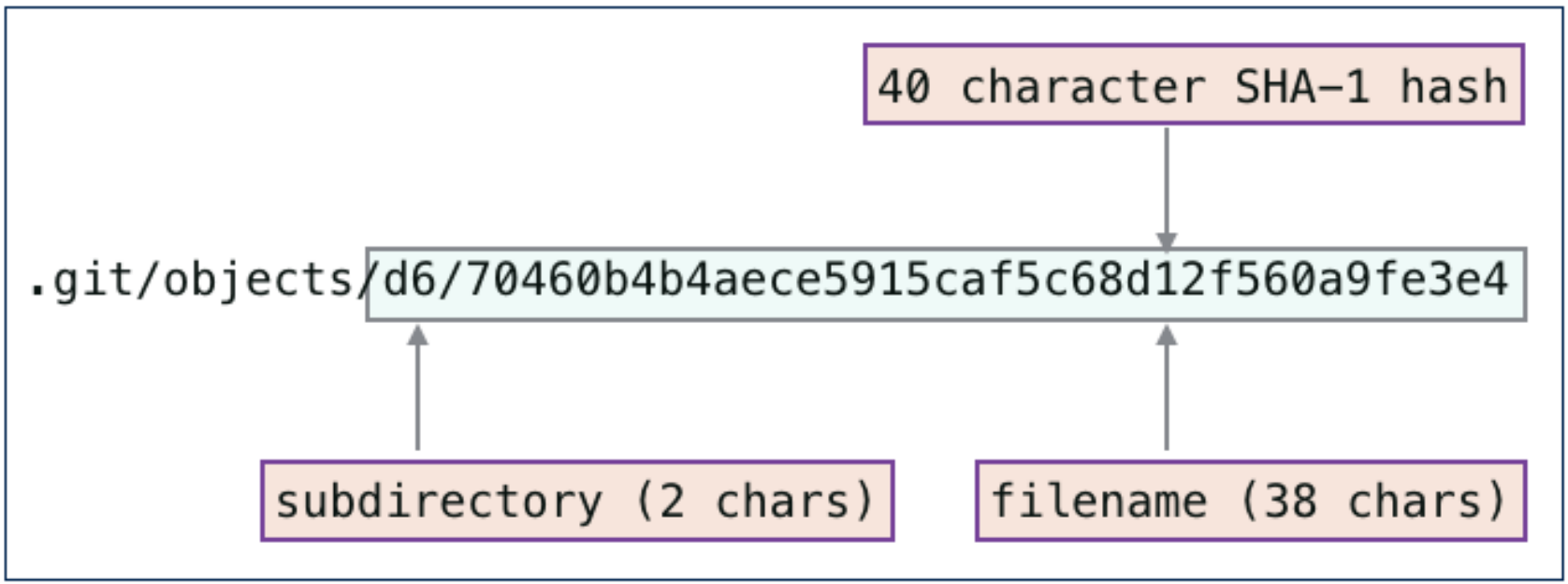
- **remotes**: *origin*
- **heads**: *master*
- **tags**: *dev-branch*

```
ls -f1 .git/refs


.
..
remotes
heads
tags
```

objects

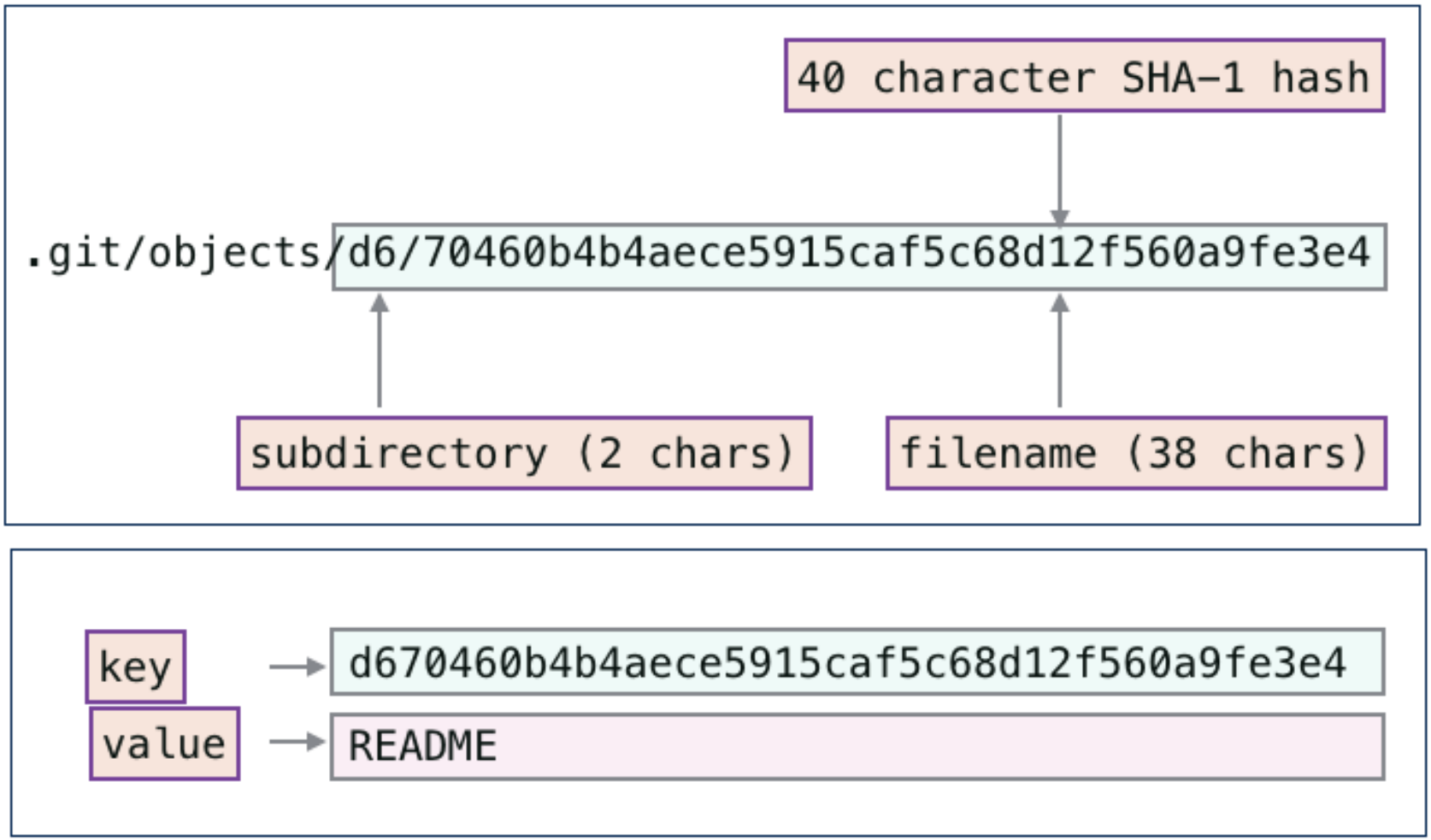
Data storage

```
find .git/objects -type f
```



objects

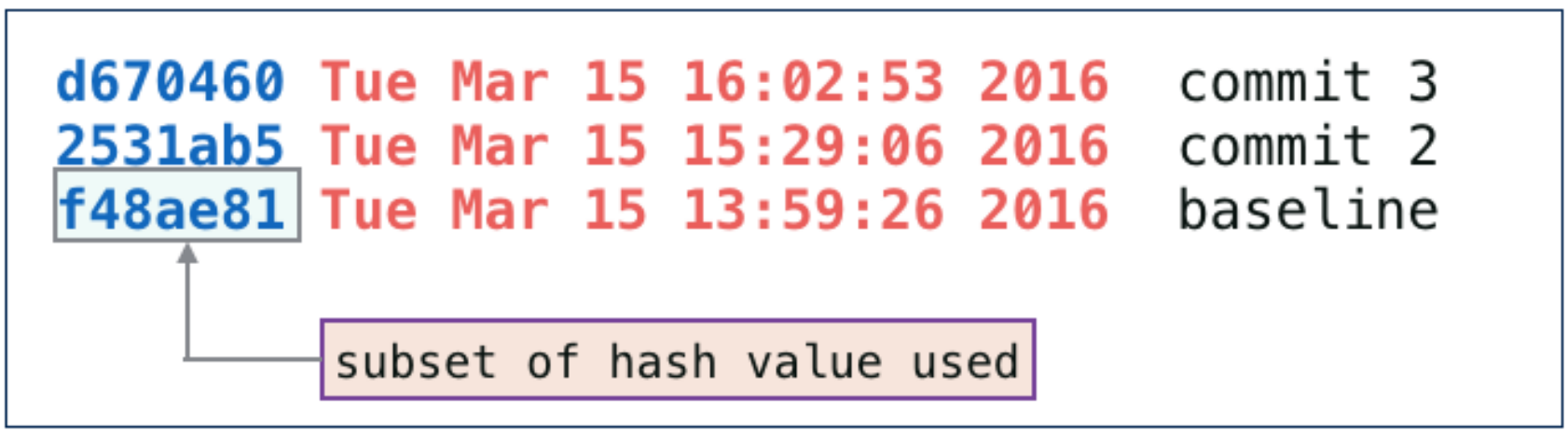
Data stored as key-value pair



objects

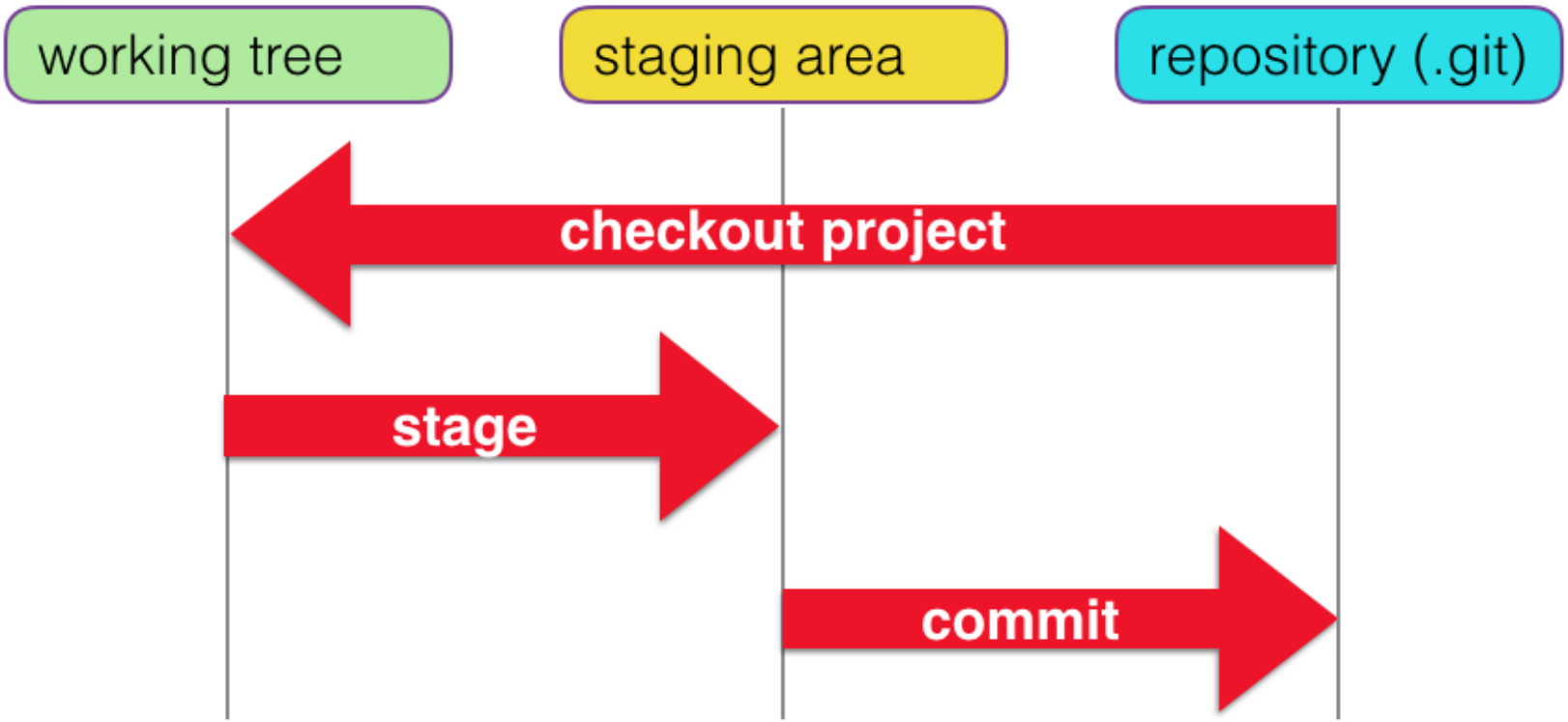
Data stored as key-value pair

- SHA-1 function generates key.
- SHA-1 (Secure Hash Algorithm)
  - Generates 160-bit, 20-byte, 40-character value
  - Pretty log printing abbreviates hash



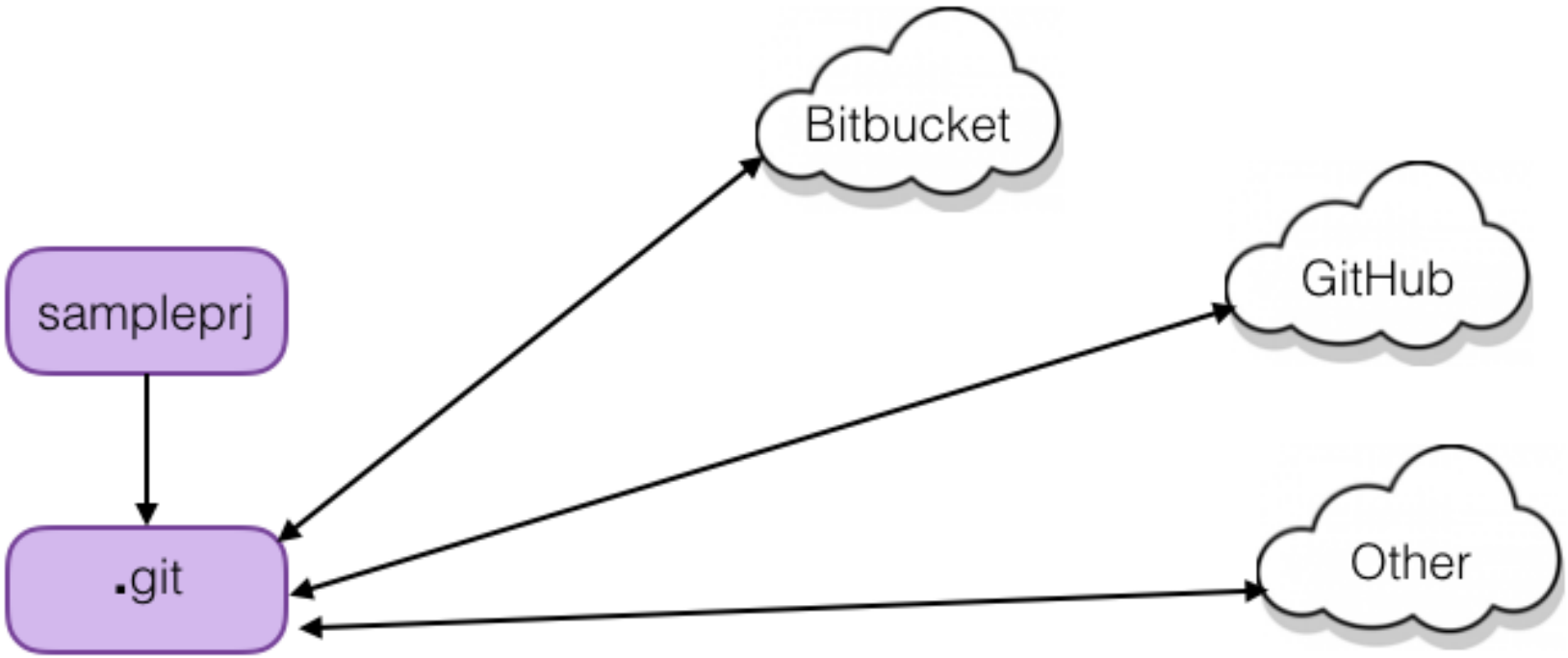
Local Git workflow

Working tree, staging area, git repository



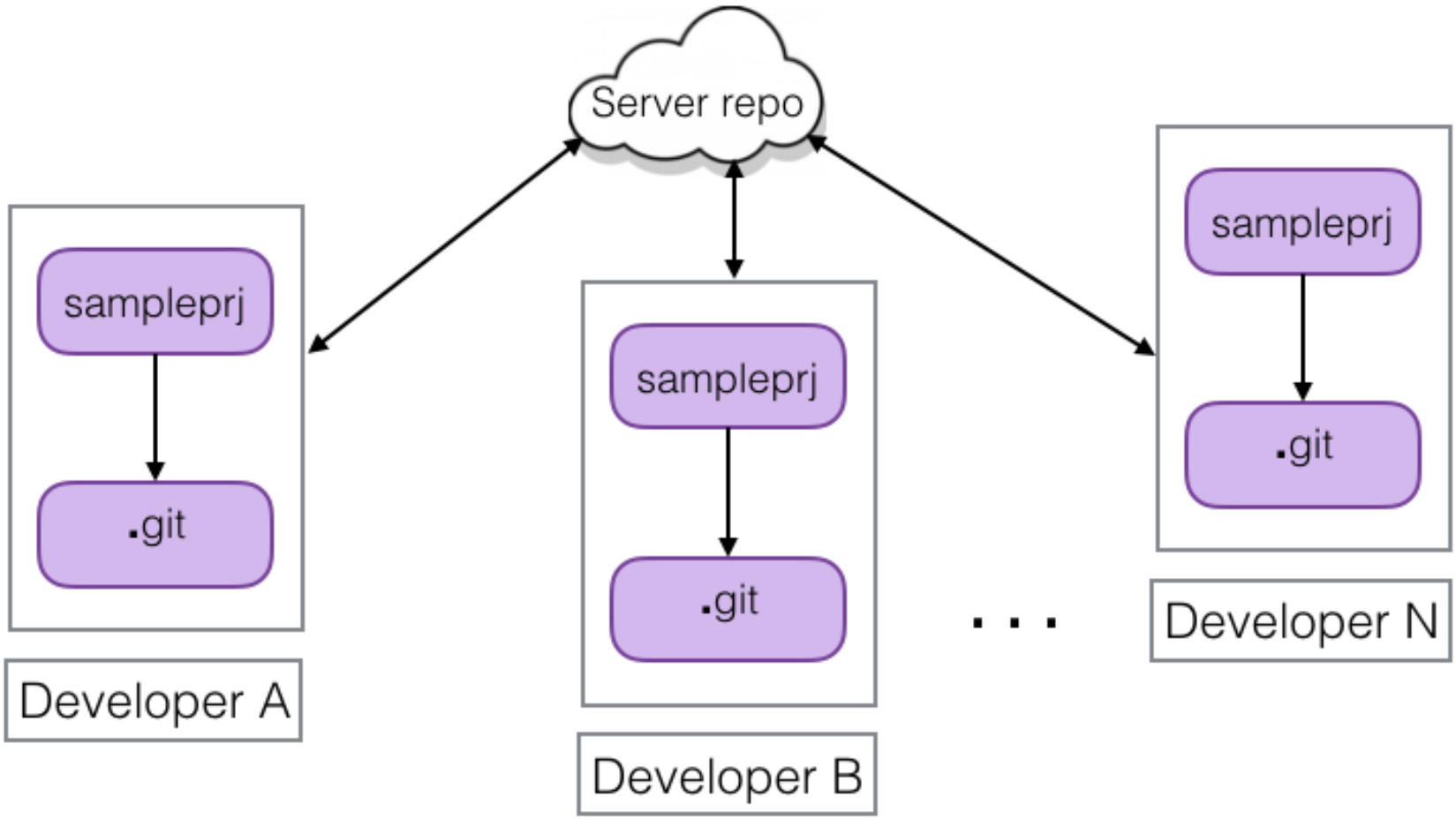
Git deployed remote servers

Track local repo on one or more cloud servers



Git deployed remote servers

Development team track local repo on one or more cloud servers



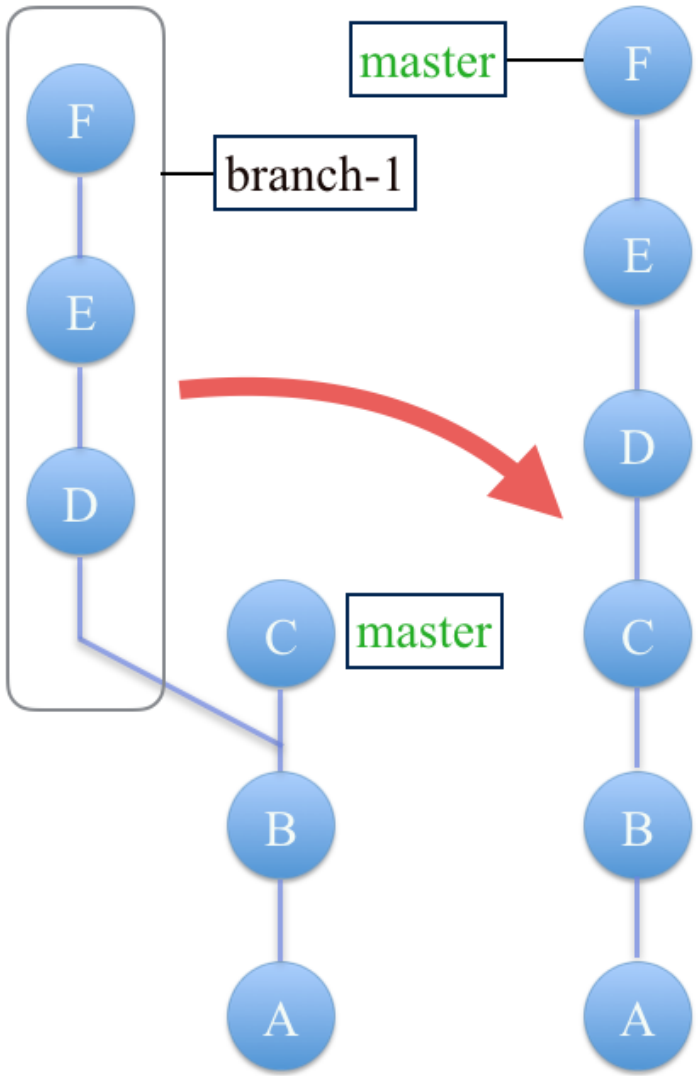
Git review

Some commands encountered in Basic Git presentation and labs

```
git init
git add README
git commit -m 'added README'
git remote add origin https://github.com/yourdomain/repo.git
git push
git pull
git status
git branch development
git checkout development
git tag -a dev0
```

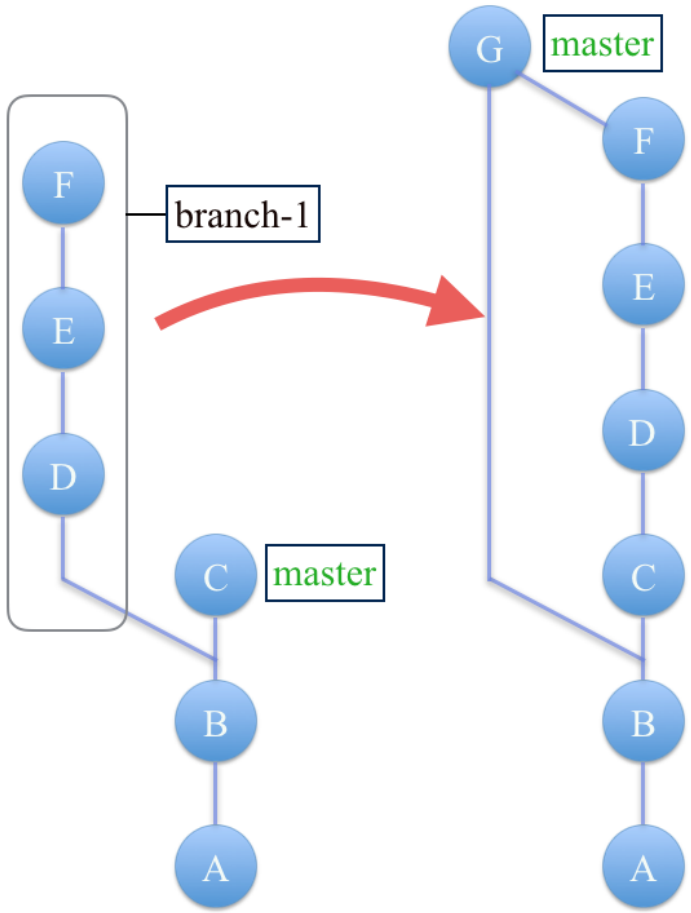
git rebase

Move branch to new base commit resulting in linear history



git merge

Integrate branch using merge resulting in master tip having two parent commits



Rewrite history

Edit last commit message

Replace last commit message:

```
git commit --amend -i 'new commit message'
```

Rewrite history

Interactive rebase to squash commits

```
a261ef4 Wed Apr 27 20:25:16 2016  commit 5
1c63d2e Wed Apr 27 20:24:54 2016  commit 4
db70a19 Wed Apr 27 20:24:31 2016  commit 3
ffe9d3a Wed Apr 27 20:24:08 2016  commit 2
7bc539b Wed Apr 27 11:20:59 2016  baseline
```

```
git rebase -i HEAD~4

pick ffe9d3a commit 2
squash db70a19 commit 3
squash 1c63d2e commit 4
squash a261ef4 commit 5
```

```
ffe9d3a Wed Apr 27 20:24:08 2016  consolidated commit
7bc539b Wed Apr 27 11:20:59 2016  baseline
```

Rewrite history

Example: interactive rebase or amend last commit

HEALTH WARNING:

Do not rewrite history of commits already pushed to remote cloud repository (public repo).

Team

Resolving conflicts

Consider the following:

- Two-developer team (A & B)
- A & B have local repos
- Shared cloud repo
- Three modules in app
  - module-A (developer A only)
  - module-B (developer B only)
  - module-Shared (A & B joint work)

Team

Resolving conflicts

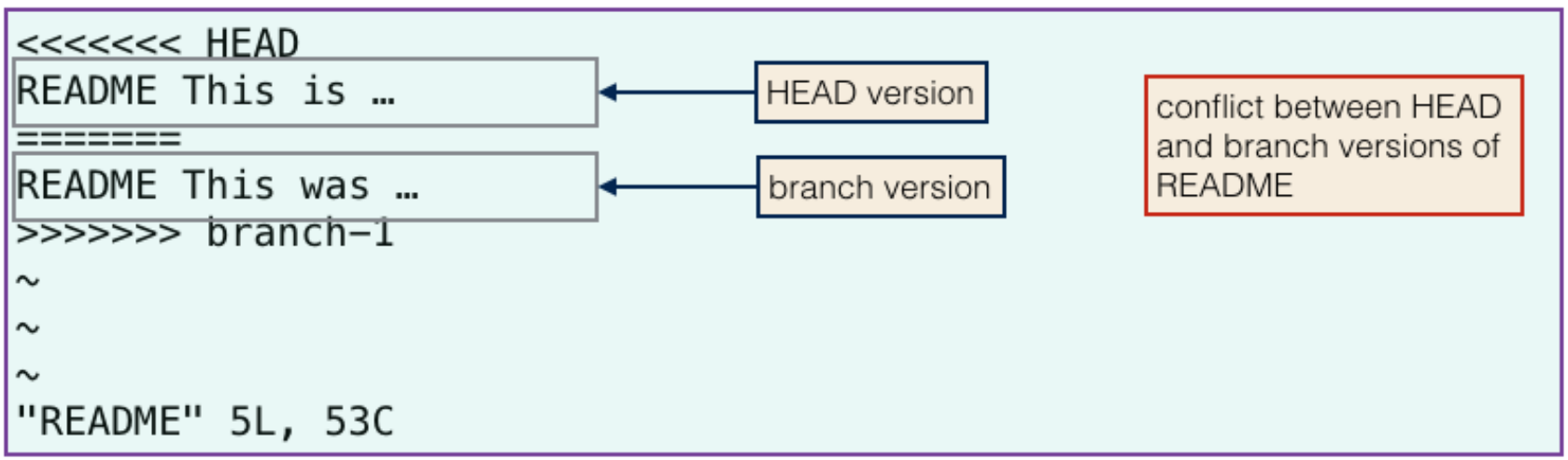
Possible scenario

- No conflict possible modules A & B
- A & B independently modify shared module:
  - A pushes module: should succeed
  - B attempts push: this will fail
    - solution: pull
    - fix conflicted files
    - save & merge

Merge conflicts

How to resolve

```
git merge branch-1
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
```




```
branch-1 & master README versions merged
```

Summary

- Four Git objects
  - Tree
  - Blob
  - Commit
  - Tag
- Relationship between project file structure and **.git** objects
  - Directory - Tree
  - File - Blob
- Git core components
  - *HEAD*
  - *index*
  - *objects*
  - *refs*

Summary

- Git: content-addressable system
  - Data stored as key-value pair
  - Key is 40 character SHA-1 hash value
- Git workflow
  - Working tree
  - Staging area
  - Repository
- Review basic commands
  - *init*
  - *add*
  - *commit*
  - *push*
  - *pull*

Summary

- Rebase
  - Move branch to new base commit
  - Linear history results
- Merge
  - Not linear history
  - Two parent commits
- Resolve conflicts
  - Using vi(m) or other text editor
- Change commit history
  - Use interactive rebase

Referenced Material

1. Pro Git by Scott Chacon & Ben Straub

`https://git-scm.com/book/en/v2` [Accessed 2016-04-27]

2. Atlassian Tutorial: Rewriting History

`https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase` [Accessed 2016-30-04]

3. Git usage statistics

`https://www.wikivs.com/wiki/Git_vs_Subversion` [Accessed 2015-03-03]