Everything You Wanted to Know About Git

Leon Jalfon
leonj@sela.co.il

Who am I?

Leon Jalfon

ALM & DevOps Consultant

SELA

Objectives

- Change the way you see and understand Git
- Don’t learn how to use Git, learn how to think in Git
- Answer the most frequently asked questions
- Show useful tips and tricks

Agenda

- Git Basics
- Git Structure
- What Branches really are?
- The Four Areas
- Common Commands
- Working with Remotes
- Merging vs Rebasing
- Most Common Workflows
- Demo using TFS 2017
- Advanced Git Overview

What is Git?

- Git is a free and open source distributed version control system designed with performance, security and flexibility in mind

- “The stupid content tracker”

  Random 3 letter combination

  Stupid, contemptible and despicable (slang)

  “Global Information Tracker"

  “Goddamn Idiotic Truckload of sh*t"

Why Should We Use Git?

- Multi-Platform
- Free and open source
- Fast and Small
- Flexible
- Secure
- Distributed
- Easy Merging (and rebasing)

Git Basics

- Git stores snapshots instead of deltas
- Each developer has a copy of the entire repository
- You can continue your work while been offline
- Branches are part of everyday development process
- Merging is central to Git (don’t be afraid of conflicts)
- Git is based on the key-value model

Meeting the SHA1

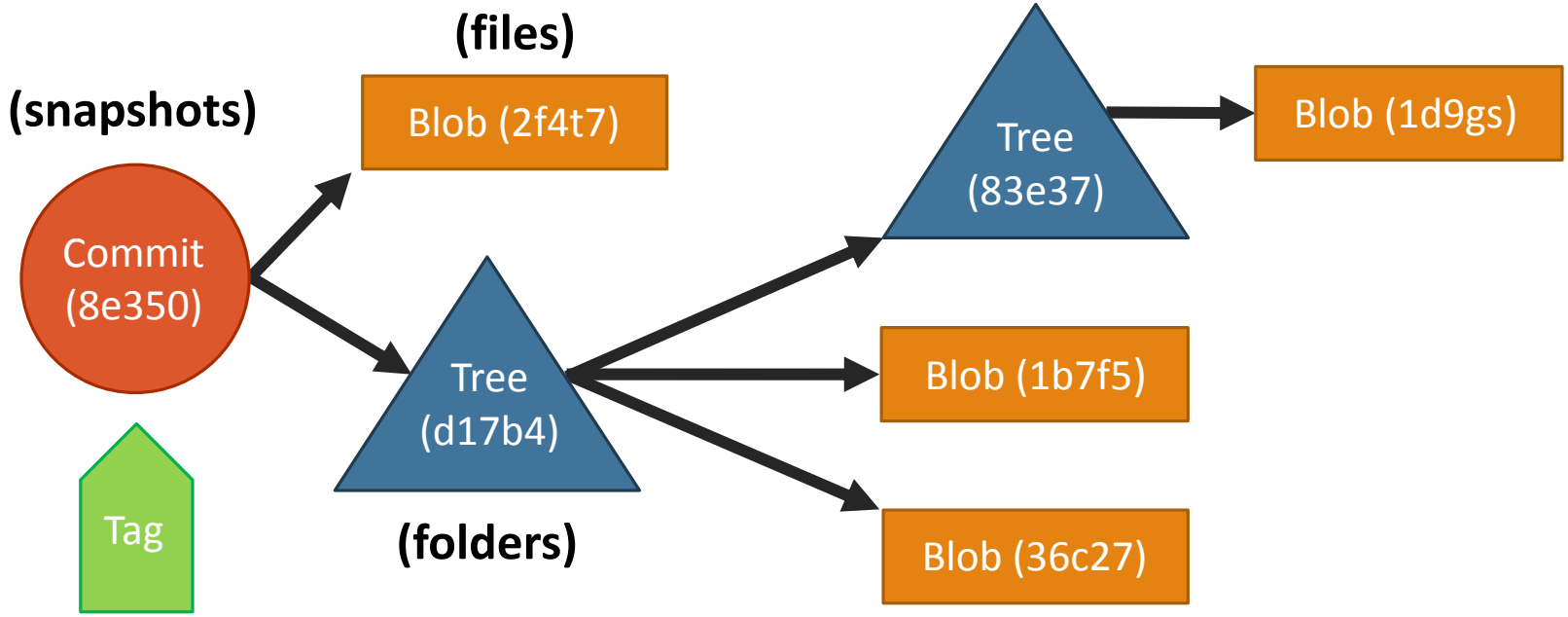
- Is a hash function that convert an long string of data into a 40 character hexadecimal number

SHA1 = **e8964**2b96685d5f22ee7044e05b9e6566e69b7a5

- Every object in Git have its own SHA1 (used as key)
- Each SHA1 is unique (or almost)
- Usually only the first 5 digits are used

$$\frac{1}{1.050.000.000}$$

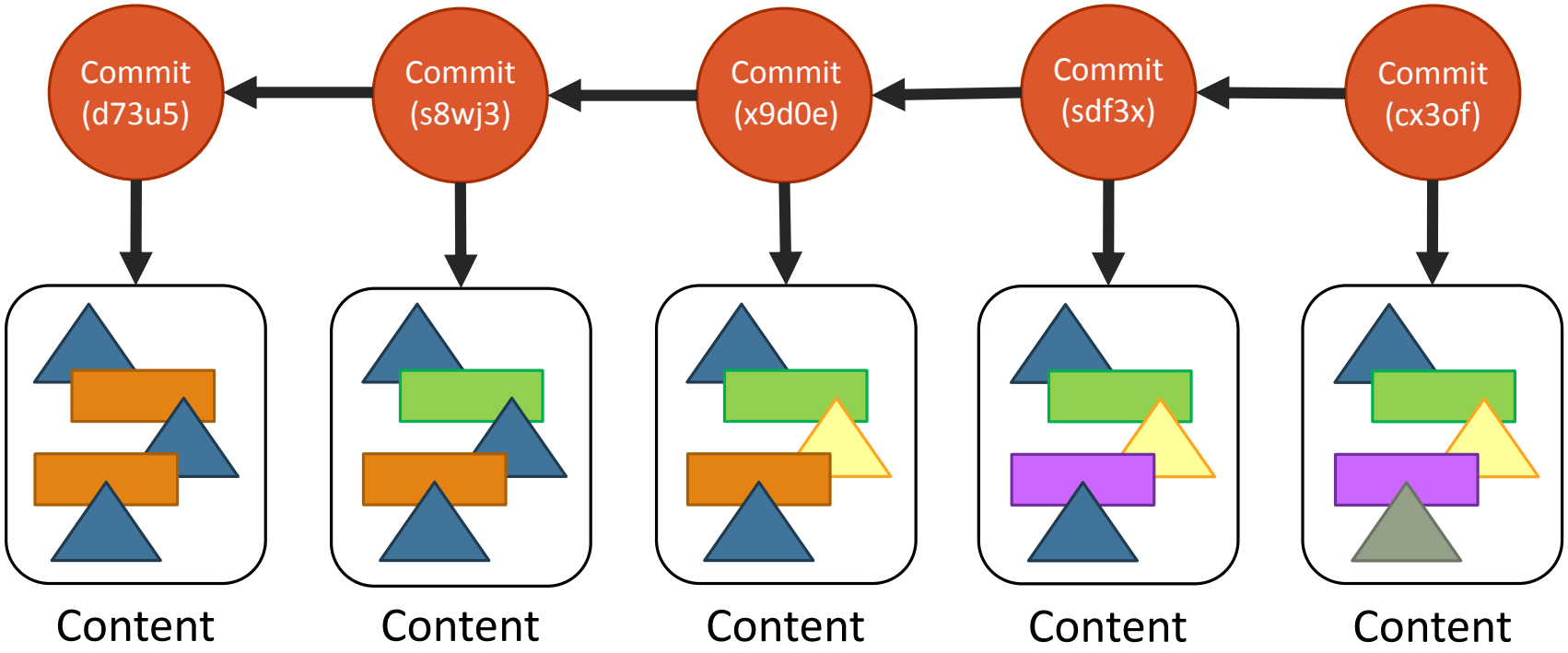
Git Structure - Objects



- A commit is a snapshot at some point in time
- A tag is a reference to a commit

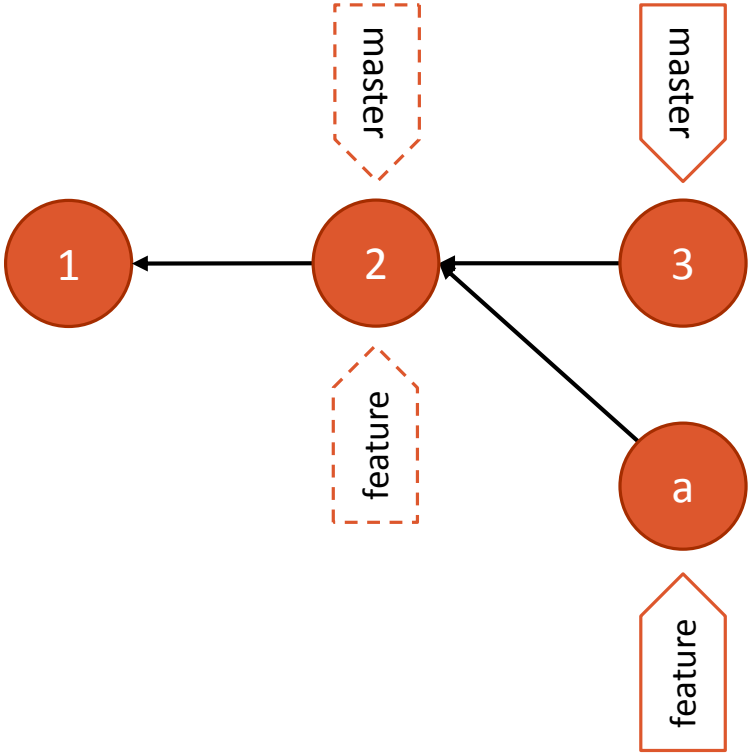
Git Structure - History

- The history is a set of interconnected commits



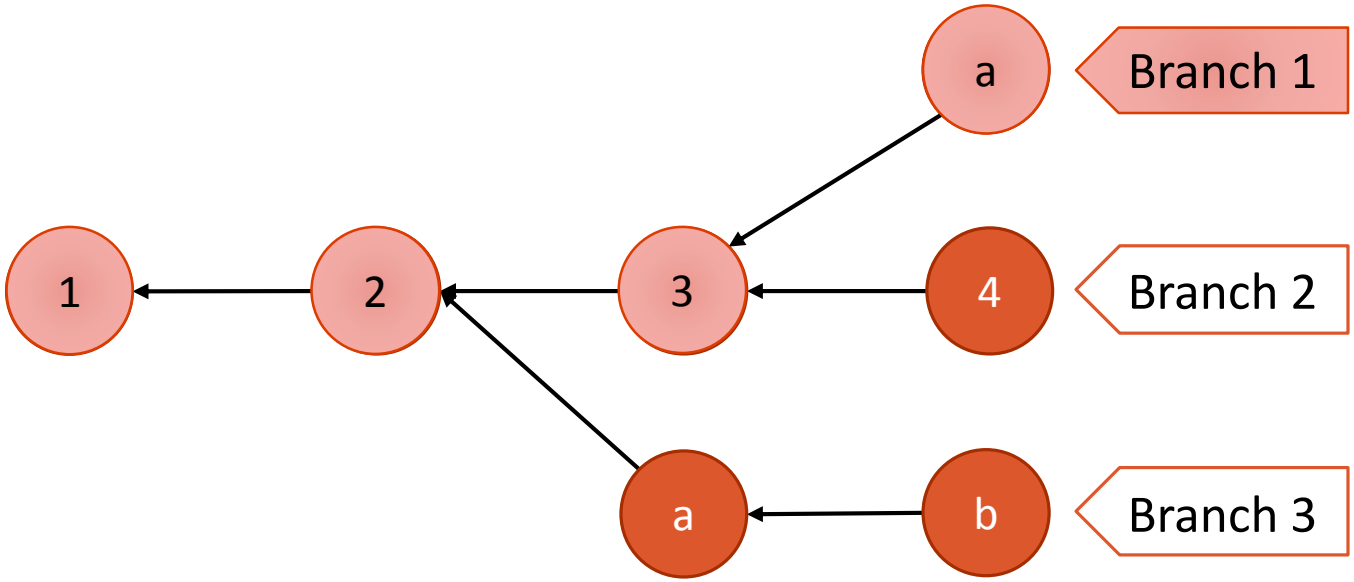
What Branches really are?

- A branch is just a reference to a commit

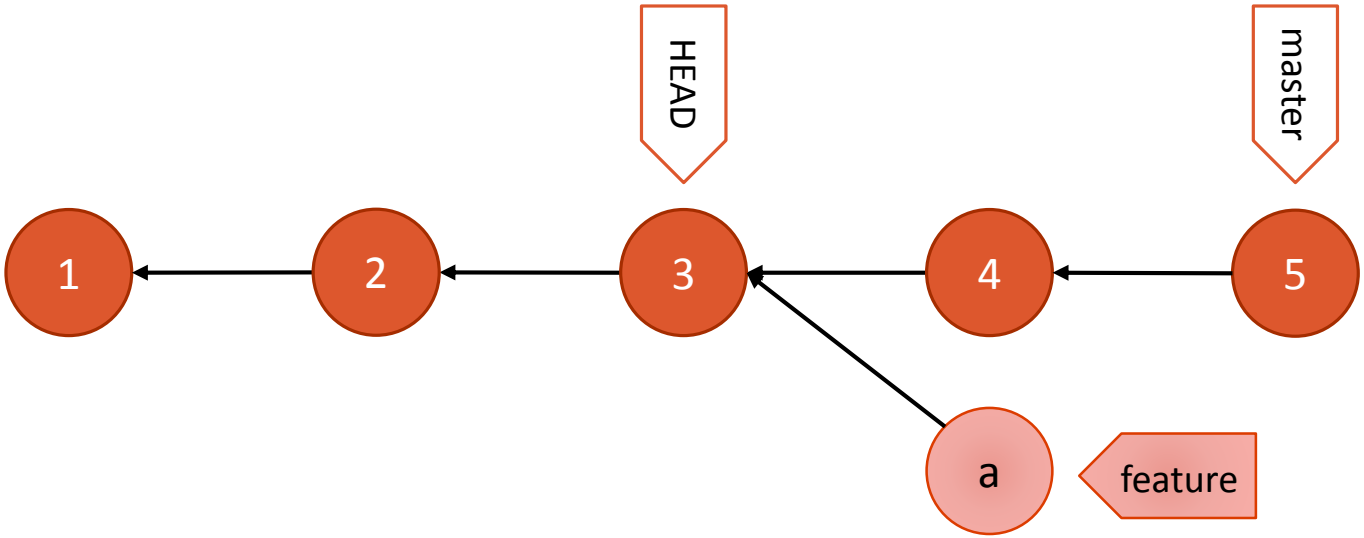


What Branches really are?

- Or rather, the entrance to the sequence…



Loosing the Head



- Happens when the HEAD is not referencing the branch tip
- Commits can't be created directly from a detached head
- To create a new commit you must create a new branch

The Four Areas

Common Commands

- Knowing a command means understanding how it affects the 4 areas
- The best way to learn the commands is by using the command line
- There are several ways to reference commits in commands:
  - Using SHA1 (partial or complete)
  - Using HEAD, branch names or tag names

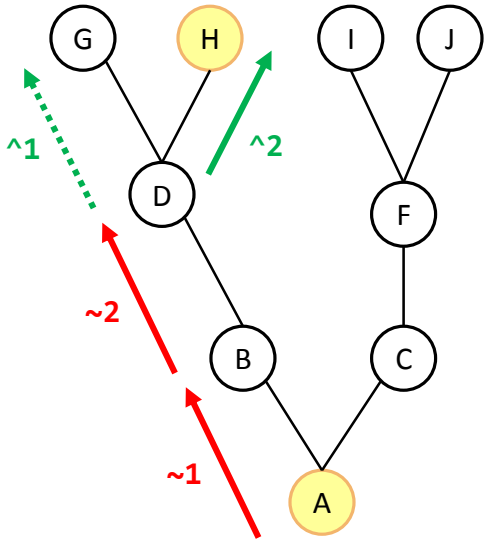
Common Commands

- Referencing from another commit

Using x~n
(n commits before x)

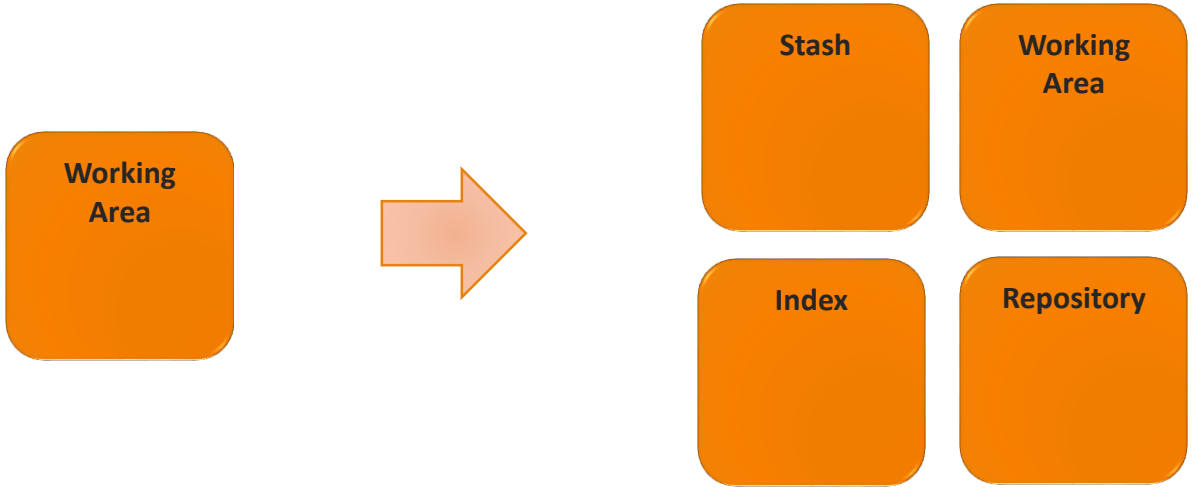
Using x^n
(n parent of x)

H	=	A	~2	^2



Common Commands

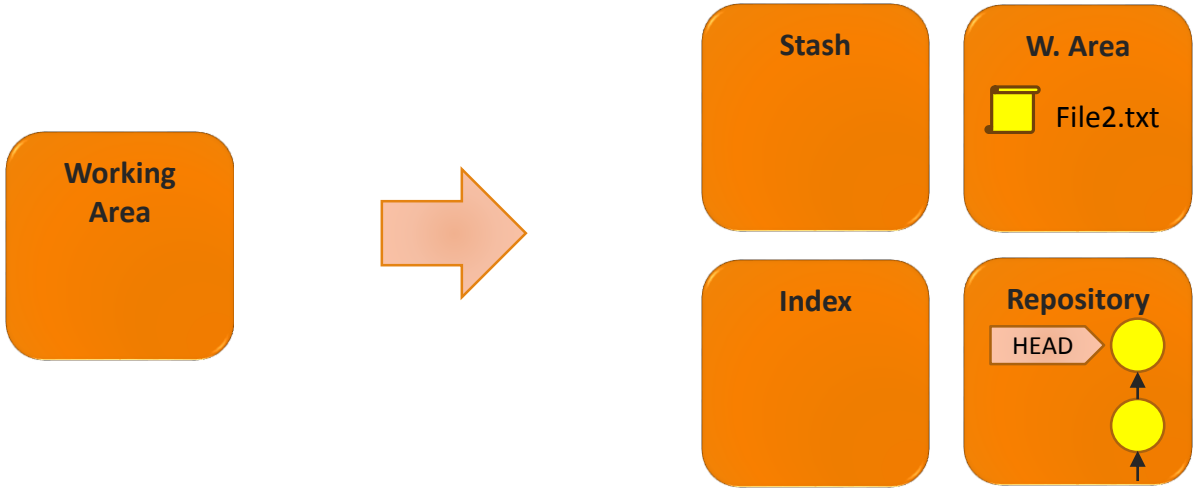
$ git init




```
$ git init

Initialized empty Git repository in
C:/Users/leonj/Desktop/MyRepo/.git/
```

Common Commands

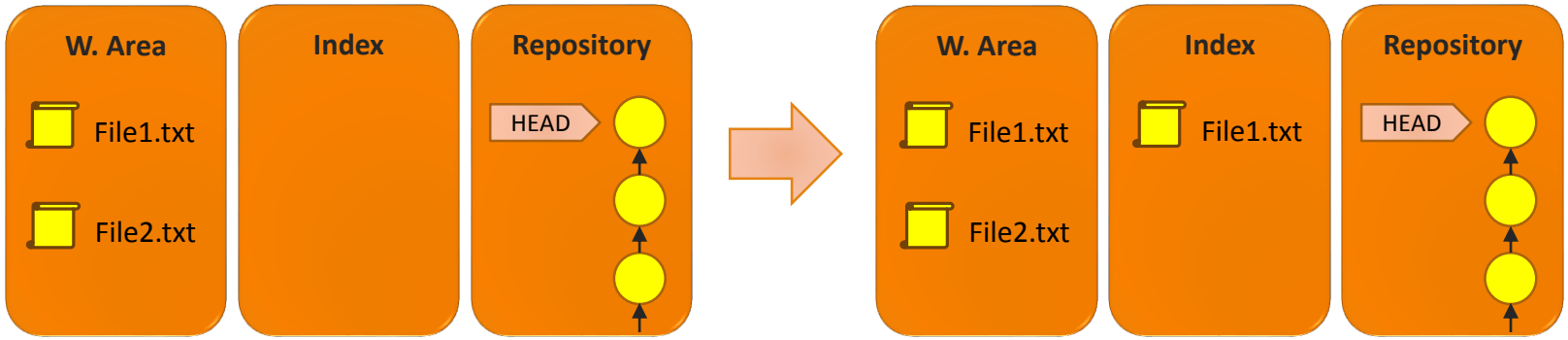
$ git clone




```
$ git clone <RepoUrl> “Folder”
  Cloning into ‘Folder’...
  Unpacking objects: 100% (19/19), done.
```

Common Commands

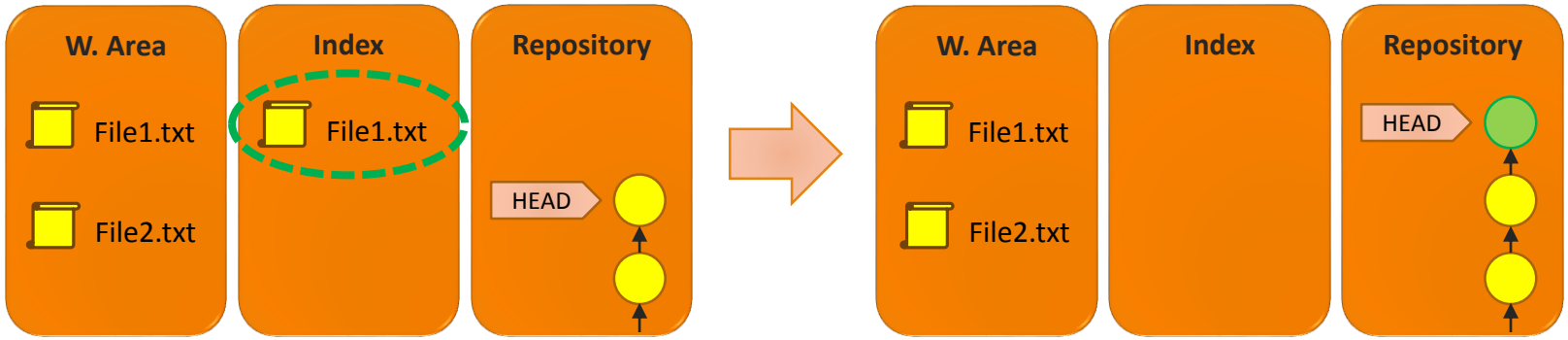
$ git add




```
$ git add "File1.txt"
$ git status
  Changes to be committed:
        modified:   File1.txt
```

Common Commands

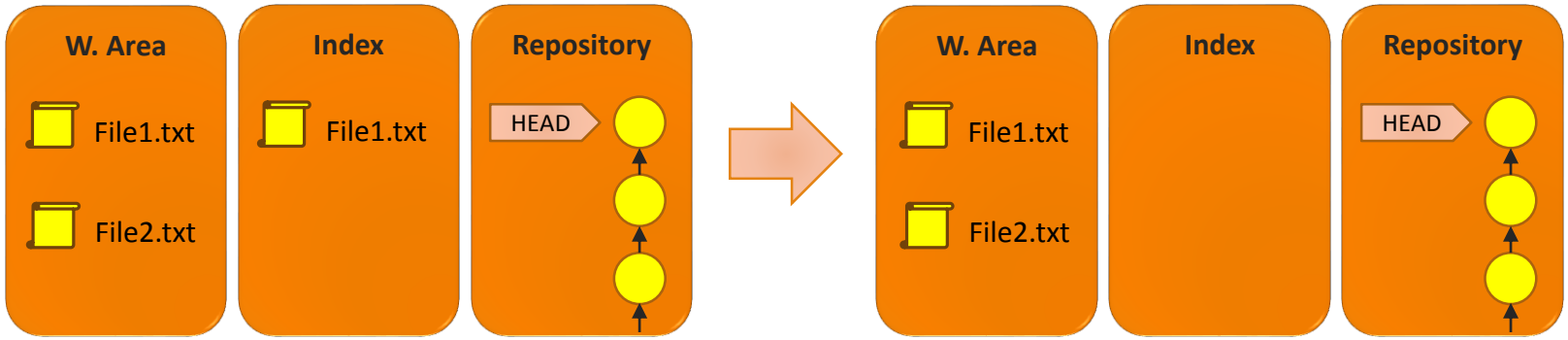
$ git commit




```
$ git commit –m “message”

  [master (root-commit) 22f1a52] message
   1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 MyFile.txt
```

Common Commands

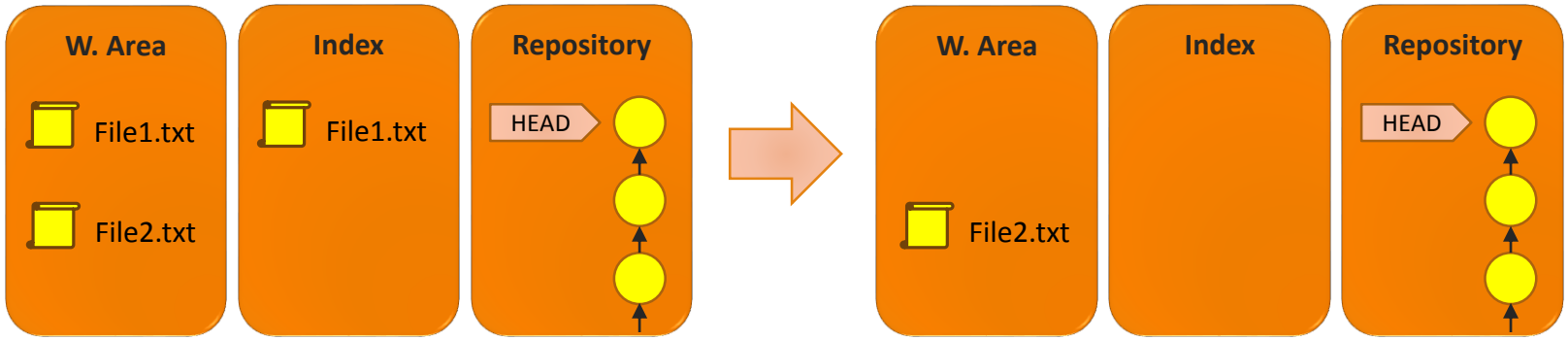
$ git rm




```
$ git rm "File1.txt"
  error: the following file has changes staged in the index
  (use --cached to keep the file, or -f to force removal)
$ git rm --cached "File1.txt"
```

Common Commands

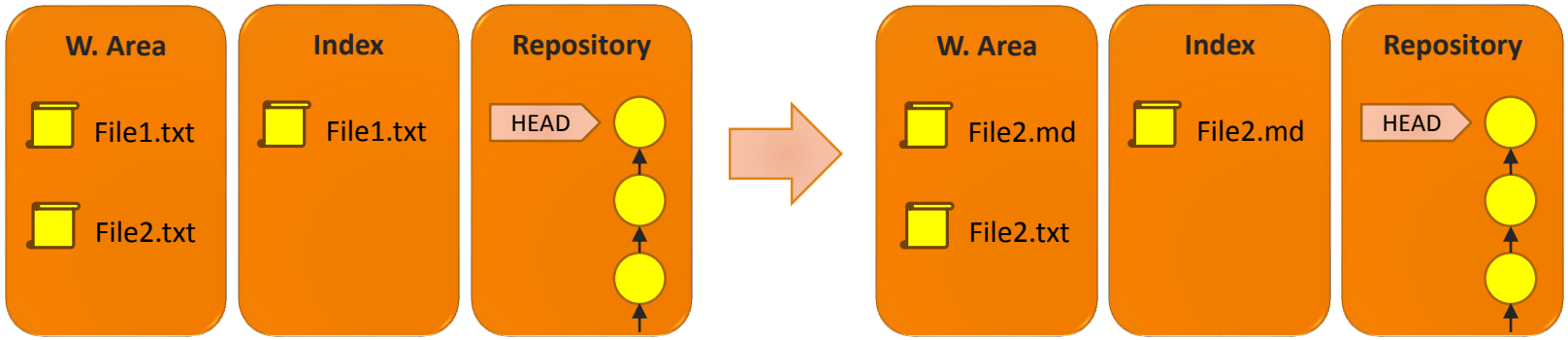
$ git rm




```
$ git rm "File1.txt"
  error: the following file has changes staged in the index
  (use --cached to keep the file, or -f to force removal)
$ git rm -f "File1.txt"
```

Common Commands

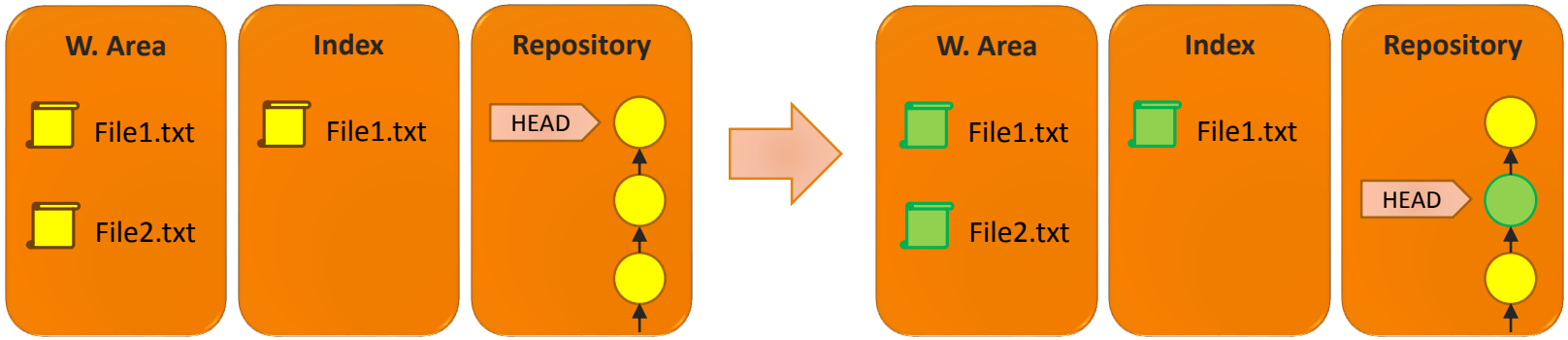
$ git mv




```
$ git mv "File1.txt" "File2.md"
$ git status
  Changes to be committed:
        renamed:   File1.txt-> File2.md
```

Common Commands

$ git checkout




```
$ git checkout HEAD~1
  Note: checking out 'HEAD~1'.
  You are in 'detached HEAD' state. You can look around...
  HEAD is now at f3d3375... <Commit Message>
```

Common Commands

$ git reset

$git reset moves the current branch, and optionally copies data from the Repository to the other areas

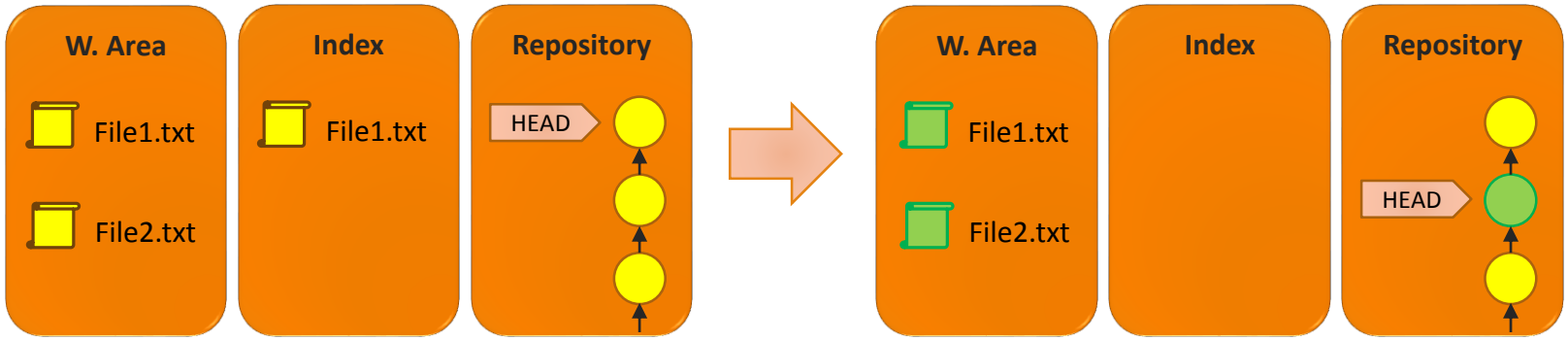
`--soft`

`--mixed`

`--hard`

Common Commands

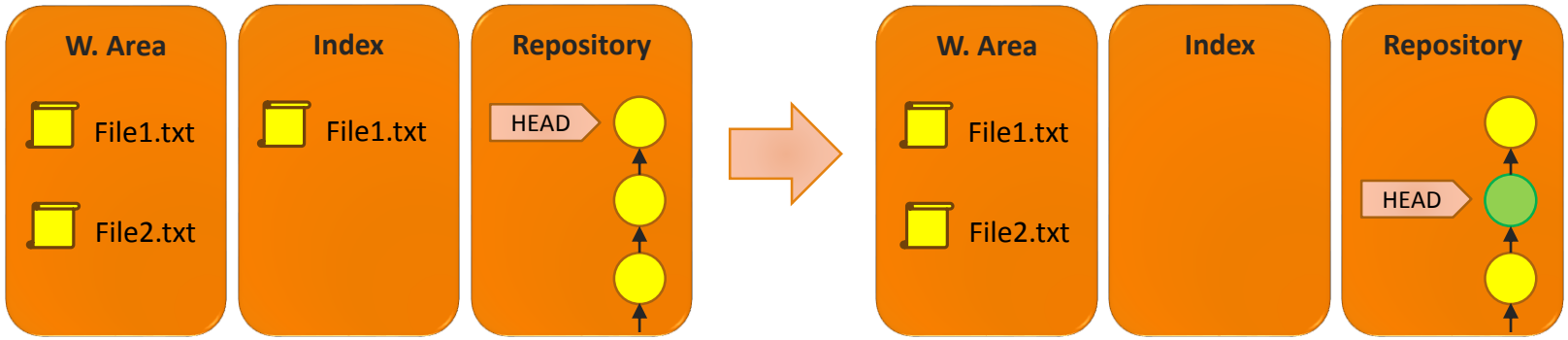
$ git reset --hard




```
$ git reset --hard HEAD~1

  HEAD is now at ec288f2 CommitMessage
```

Common Commands

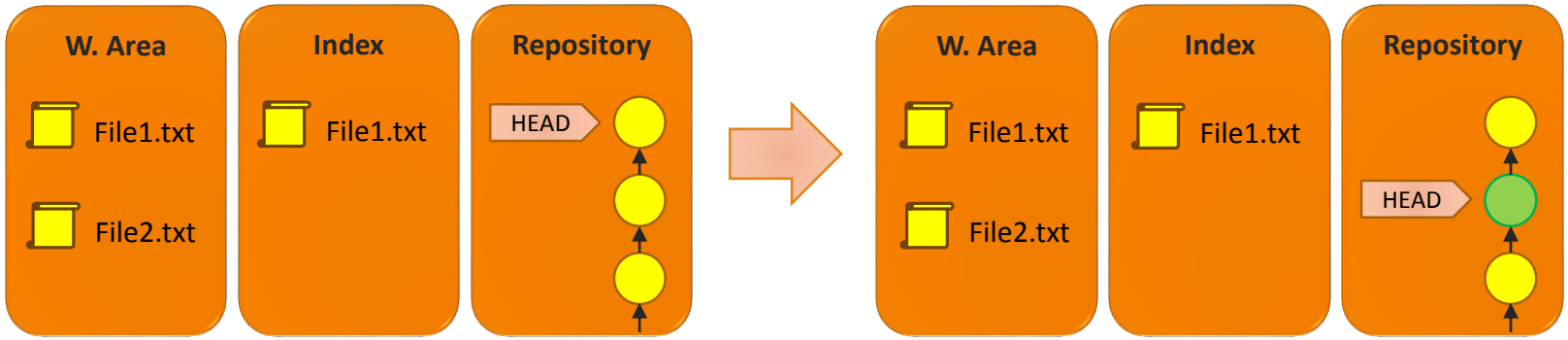
$ git reset --mixed




```
$ git reset --mixed HEAD~1

  HEAD is now at ec288f2 CommitMessage
```

Common Commands

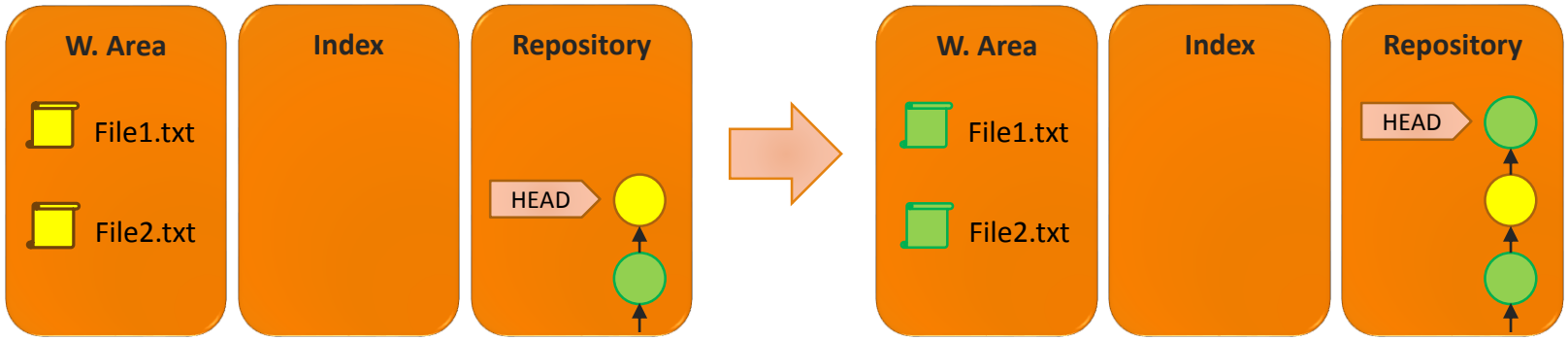
$ git reset --soft




```
$ git reset --soft HEAD~1

  HEAD is now at ec288f2 CommitMessage
```

Common Commands

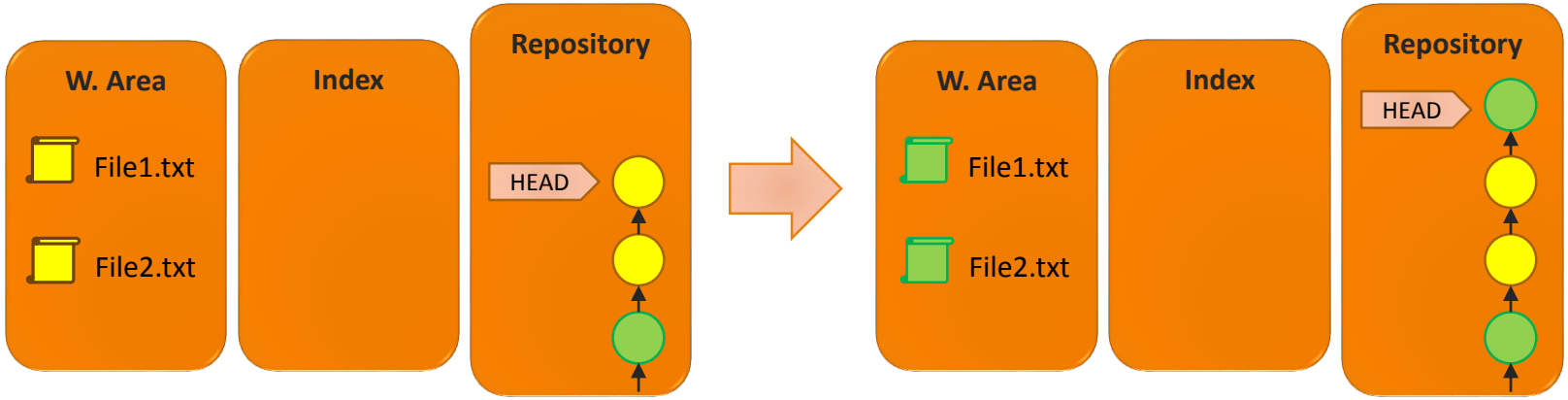
$ git revert




```
$ git revert HEAD

  Revert "commitMessage"

  This reverts commit 6b32e600dff05e6b27e0e42eeb829ee735134336
```

Common Commands

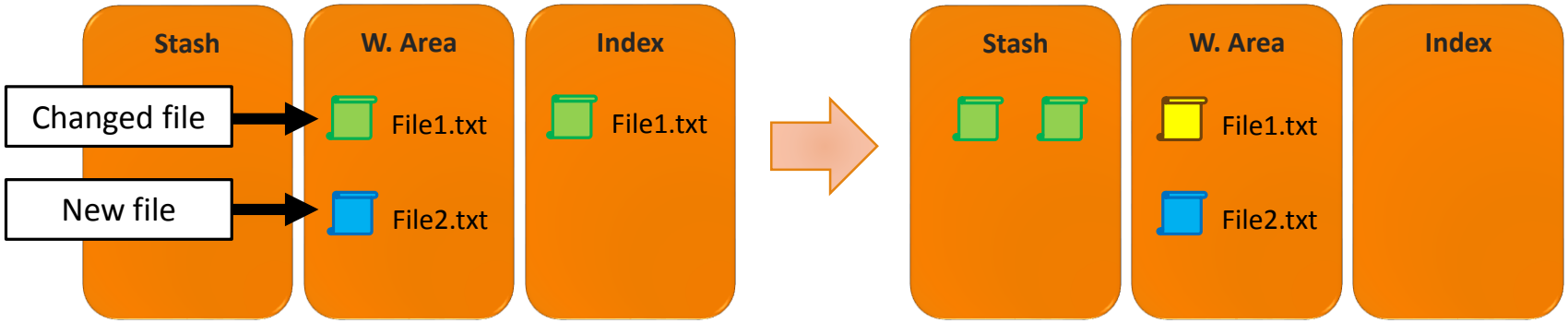
$ git cherry-pick




```
$ git cherry-pick HEAD~3

  [master c85d13a] commit
   Date: Sat Mar 11 18:38:24 2017 +0200
   1 file changed, 1 insertion(+)
```

Common Commands

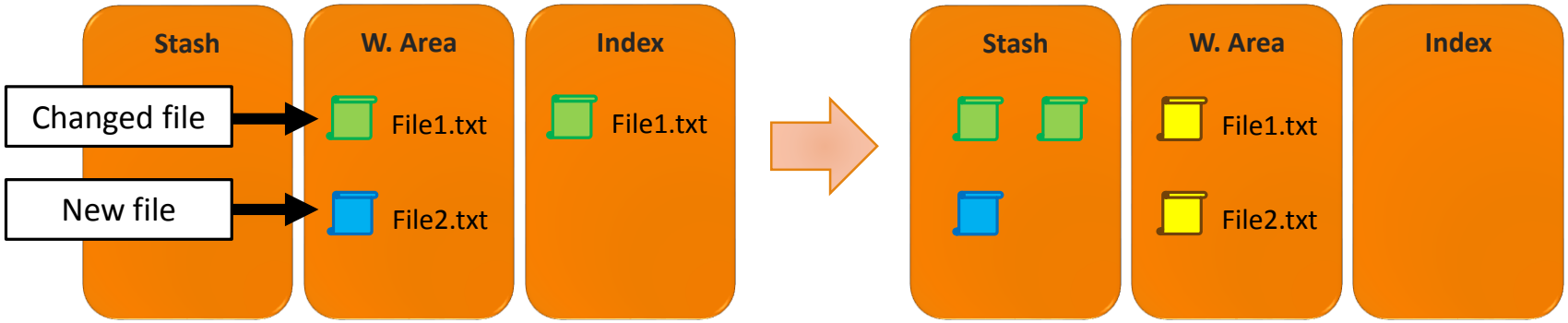
$ git stash




```
$ git stash

  Saved working directory and index state WIP on master:
  c85d13a commit
  HEAD is now at c85d13a commit
```

Common Commands

$ git stash




```
$ git stash --include-untracked

  Saved working directory and index state WIP on master:
  c85d13a commit
  HEAD is now at c85d13a commit
```

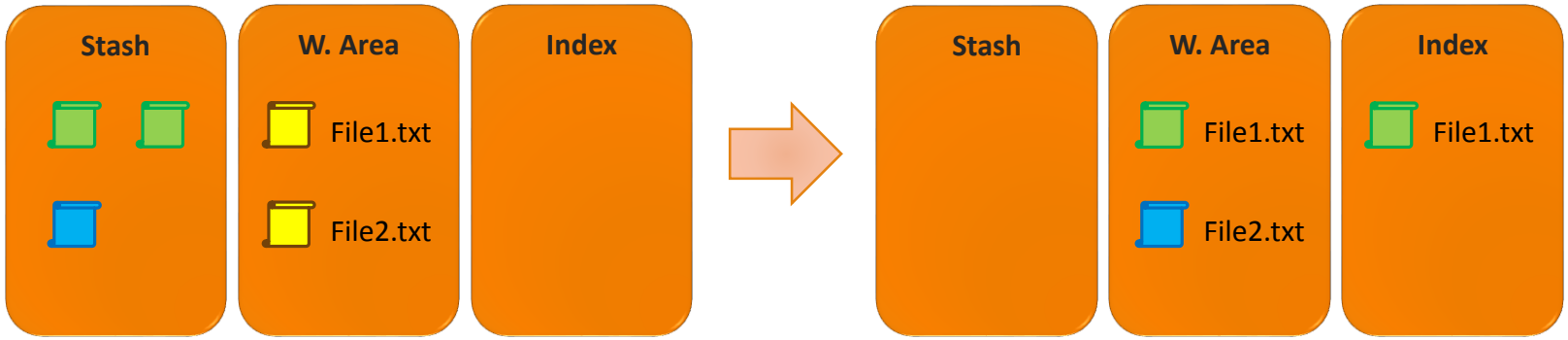
Common Commands

$ git stash list


```
$ git stash list

  stash@{0}: WIP on master: c85d13a commit
  stash@{1}: WIP on master: c85d13a commit
```

Common Commands

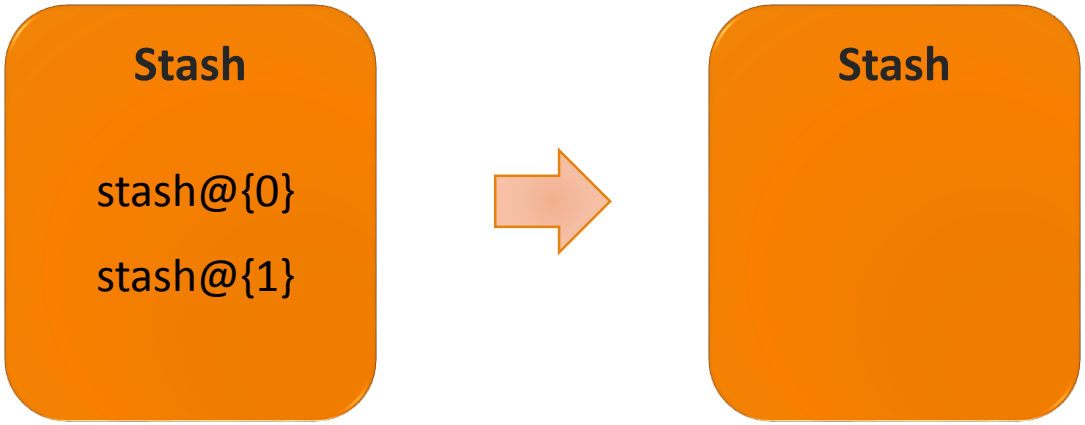
$ git stash apply




```
$ git stash apply stash@{0}

  On branch master
  Changes to be committed:
  ...
```

Common Commands

$ git stash clear



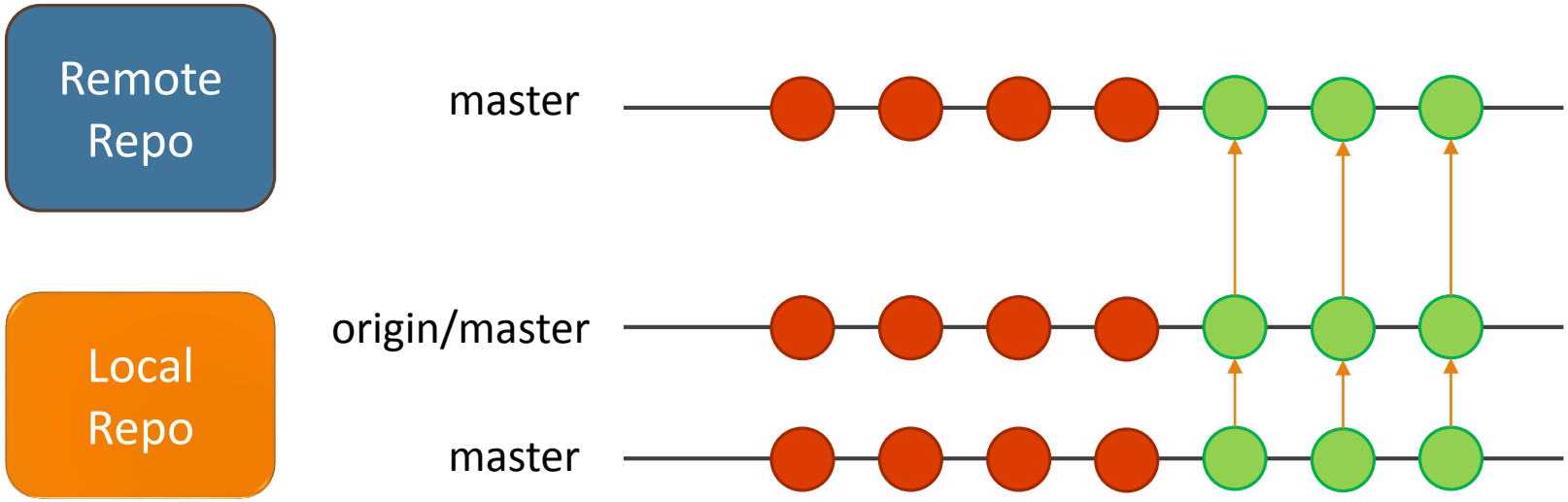

```
$ git stash clear
```

Working with Remotes

- Remote repositories are versions of the project that are hosted in another place
- You can associate several remote connections
- The default remote connection is called “origin”
- There is a hidden branch in the local repository for each branch of the remote repository (used for synchronize with the remote repository)

Working with Remotes

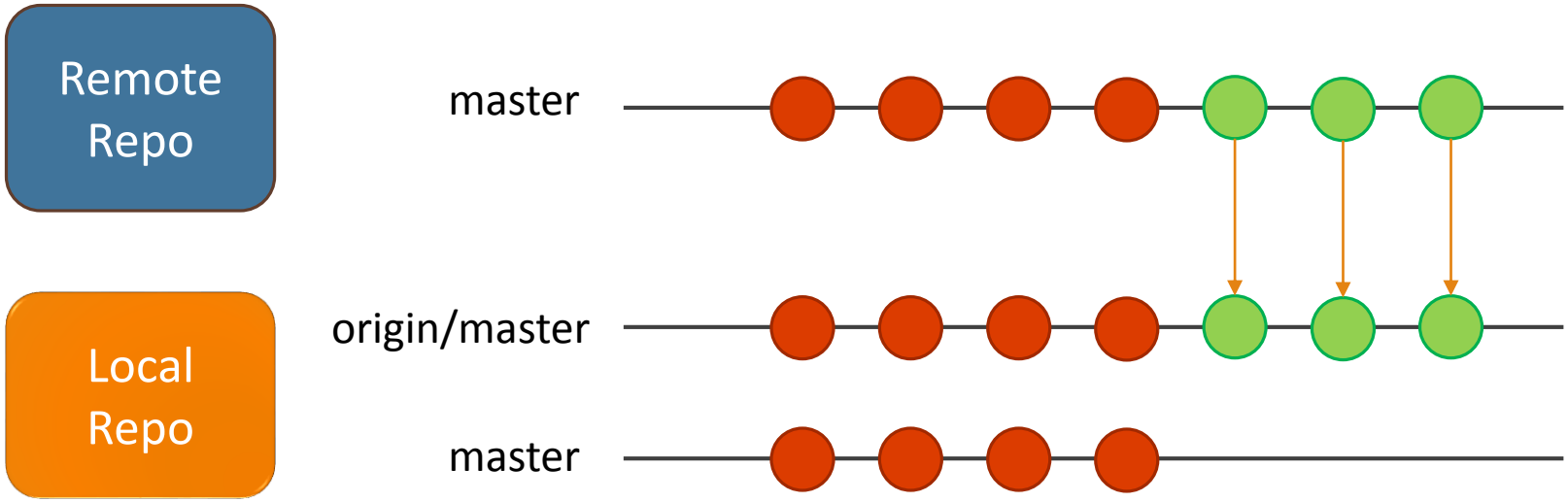
$ git push




```
$ git push origin master
   Counting objects: 3, done.
   Writing objects: 100% (3/3), 243 bytes | 0 bytes/s, done.
   52938bc..a330c52  master -> master
```

Working with Remotes

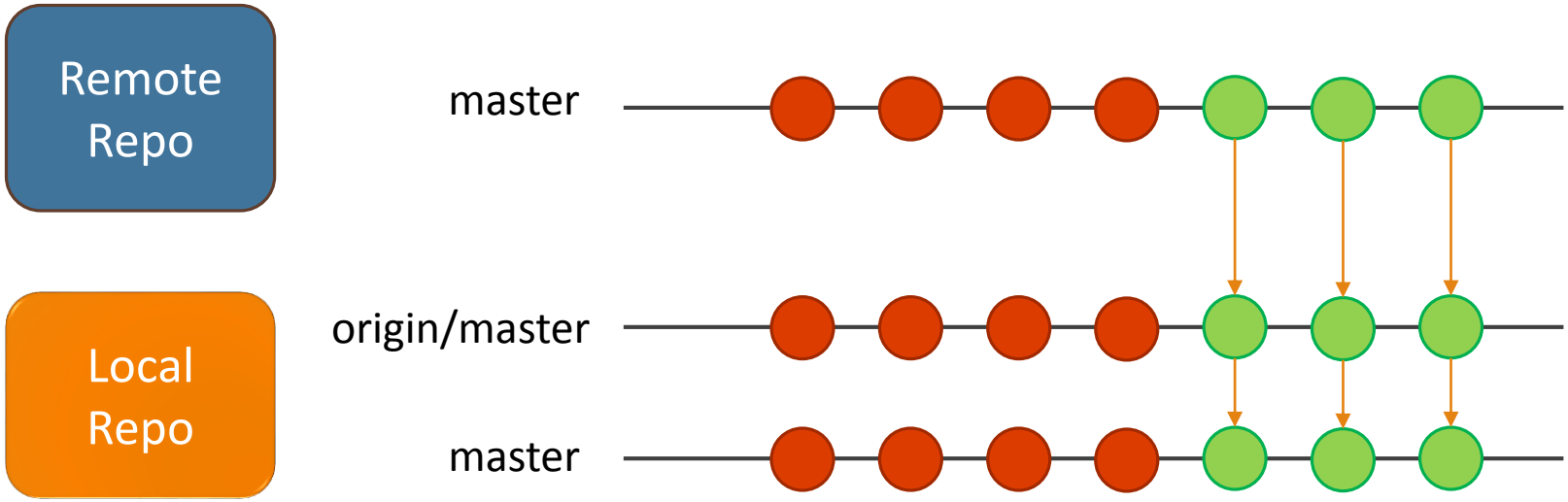
$ git fetch




```
$ git fetch origin master
   From https://leonj.visualstudio.com/DefaultCollection/Repo/_git/Test
    * branch            master     -> FETCH_HEAD
      14a2e05..a2e111a  master     -> origin/master
```

Working with Remotes

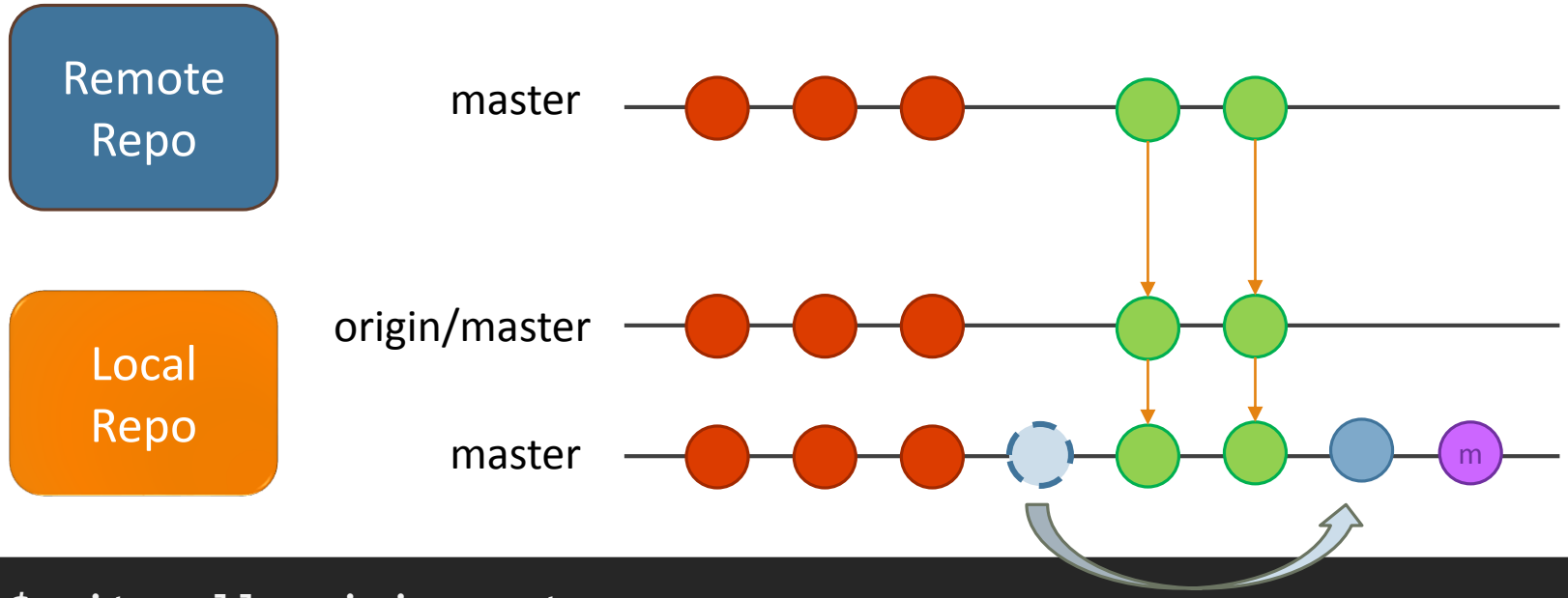
$ git pull (fetch + merge)




```
$ git pull origin master
    * branch            master     -> FETCH_HEAD
   Updating 60e69bb..a2e111a
   Fast-forward
```

Working with Remotes

$ git pull (fetch + merge)



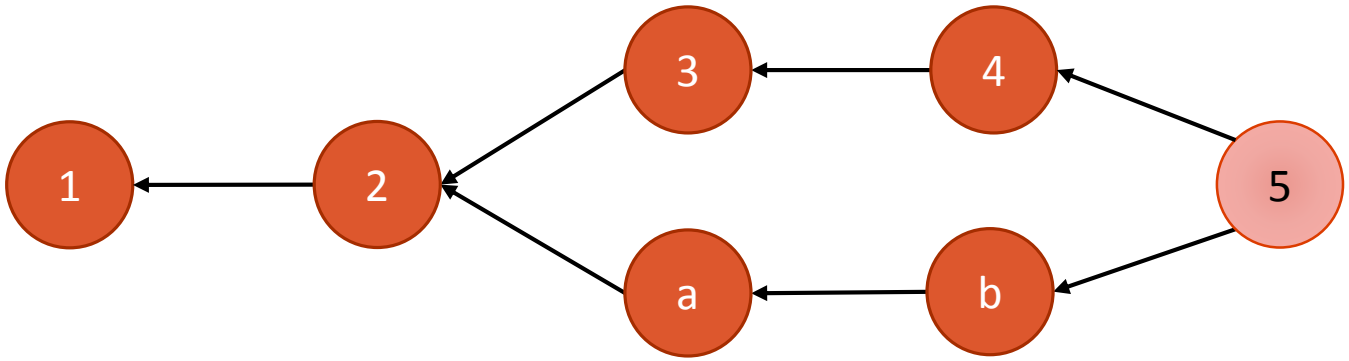

```
$ git pull origin master
    a2e111a..0532902  master     -> origin/master
   Auto-merging File.txt
   Merge made by the 'recursive' strategy.
```

Merging VS Rebasing

- What is Merge?
- What is Rebase?
- When to Merge and when to Rebase?

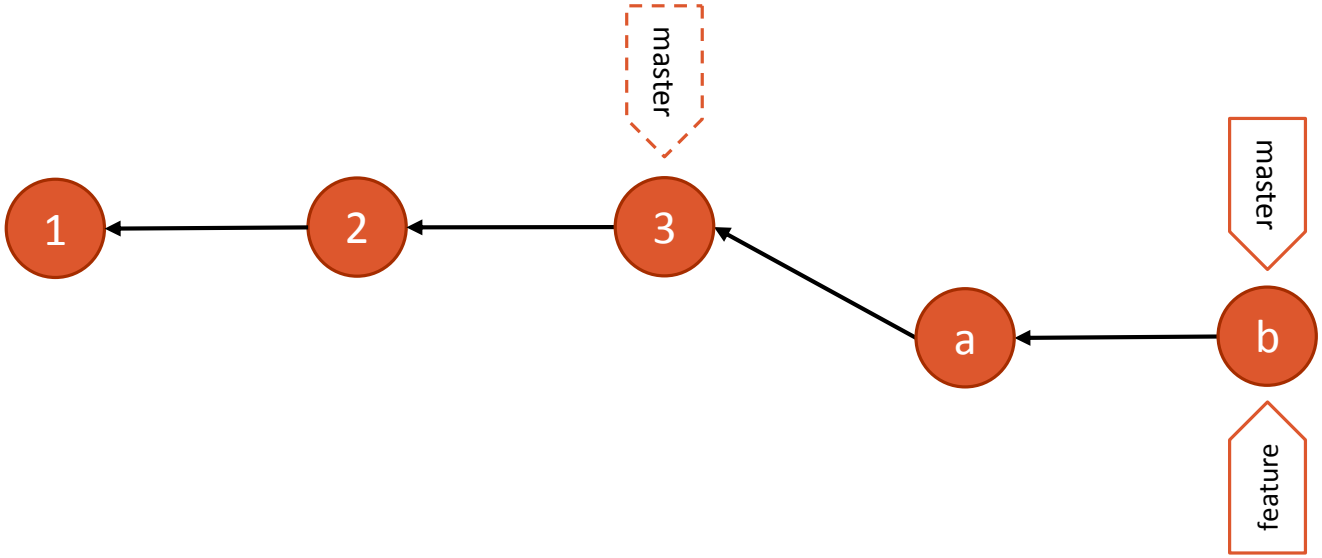
Merging
Simplified

- Merge is just a commit with two parents



Merging

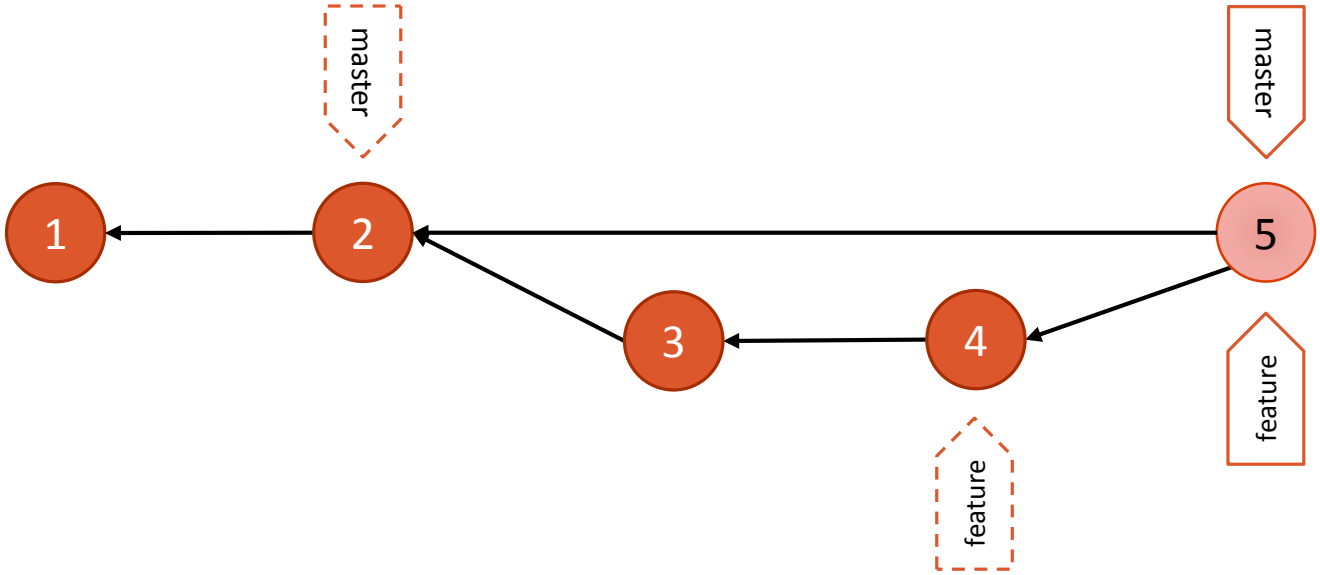
$ git merge (Fast-Forward)




```
$ git checkout master
$ git merge feature
  Updating e06ff55..b1b9382
  Fast-forward
```

Merging

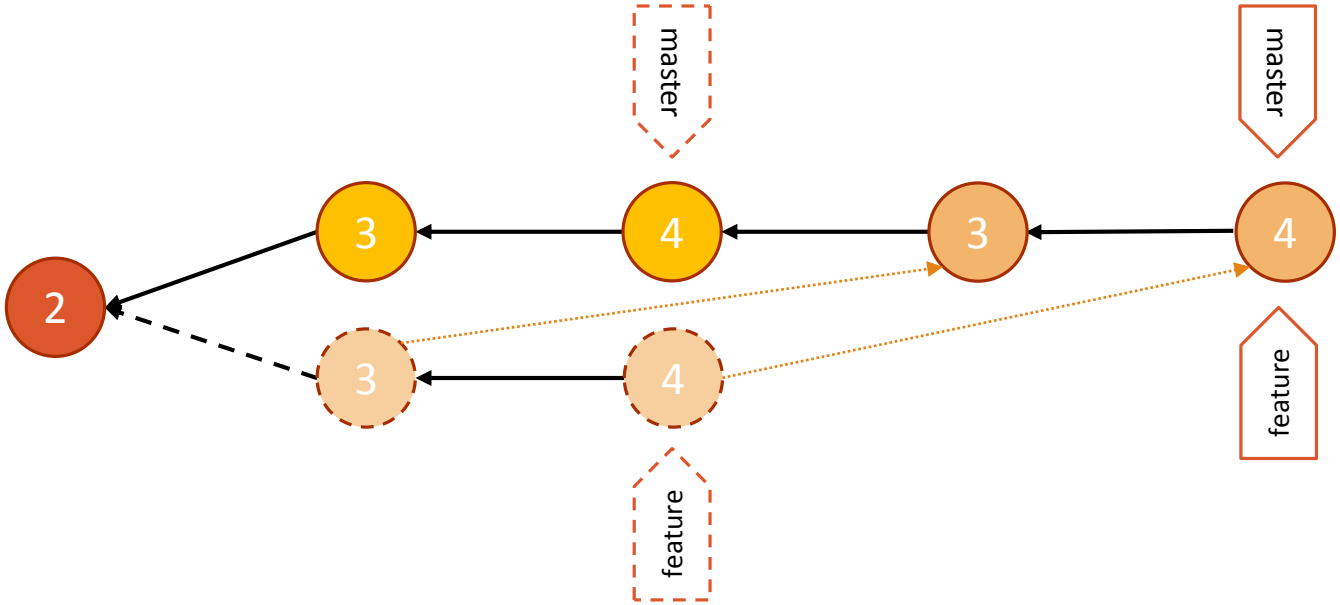
$ git merge --no-ff




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

Rebasing

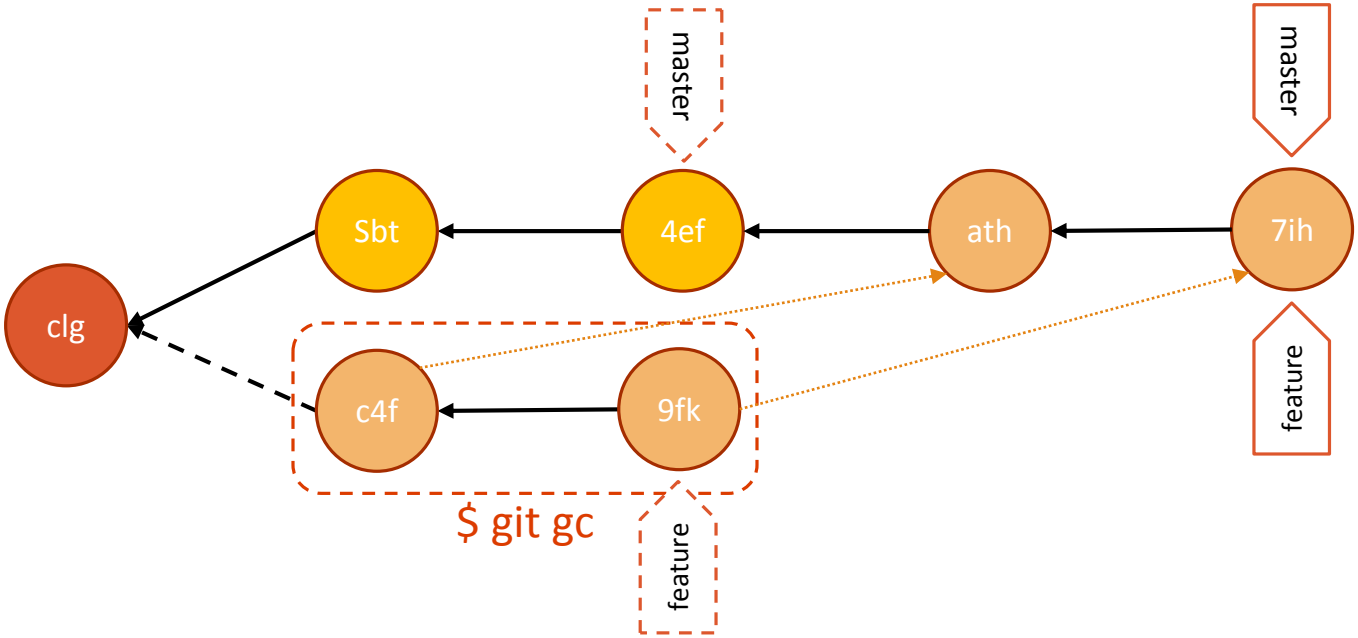
How it looks?




```
$ git checkout master
$ git rebase feature
   First, rewinding head to replay your work on top of it...
   Fast-forwarded master to feature.
```

Rebasing

What really happens?

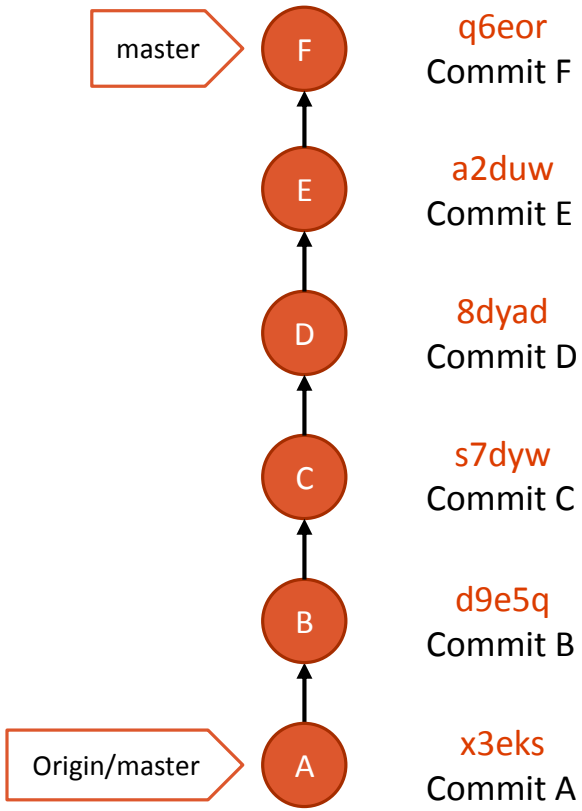



```
$ git checkout master
$ git rebase feature
   First, rewinding head to replay your work on top of it...
   Fast-forwarded master to feature.
```

Rebasing

$ git rebase -i

```
$ git log --oneline --decorate --graph

* q6eor (HEAD -> master) Commit F
* a2duw Commit E
* 8dyad Commit D
* s7dyw Commit C
* d9e5q Commit B
* x3eks (origin/master) Commit A

$ git rebase -i
```

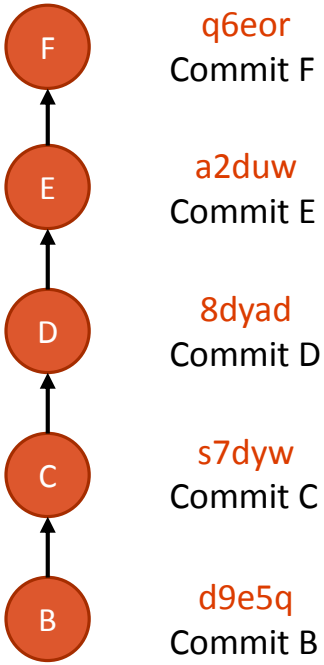


Rebasing

$ git rebase -i

```
pick   d9e5q   Commit B
pick   s7dyw   Commit C
pick   8dyad   Commit D
pick   a2duw   Commit E
pick   q6eor   Commit F

# Rebase x3eks..q6or7 onto x3eks...
#
# Commands:
# ...
# ...
```

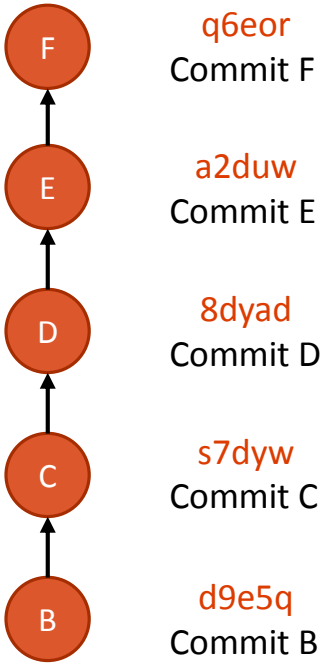
```
[1] <rge/git-rebase-todo [+][gitrebase][Git(master)]    All
:wq
```



Rebasing

$ git rebase -i

```
pick   d9e5q   Commit B
squash s7dyw   Commit C
squash 8dyad   Commit D
pick   q6eor   Commit F
reword a2duw   Commit E

# Rebase x3eks..q6or7 onto x3eks...
#
# Commands:
# ...
# ...

[1] <rge/git-rebase-todo [+][gitrebase][Git(master)]    All
:wq
```

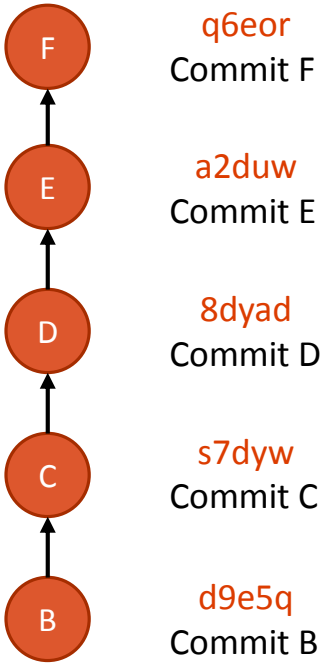


Rebasing

$ git rebase -i

```
Pick     d9e5q   Commit B
Squash   s7dyw   Commit C
Squash   8dyad   Commit D
Pick     q6eor   Commit F
Reword   a2duw   Commit E
```

```
# This is a combination of 2 commits.
# The first commit's message is:

Commit B

# This is the 2nd commit message:

Commit C

# This is the 2nd commit message:

Commit D

[1] <rge/git-rebase-todo [+][gitrebase][Git(master)]    All
:wq
```

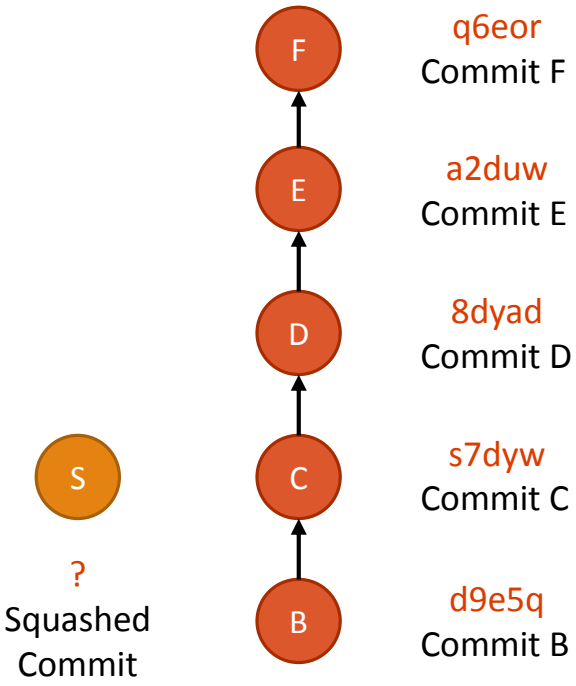


Rebasing

$ git rebase -i

```
Pick     d9e5q   Commit B
Squash   s7dyw   Commit C
Squash   8dyad   Commit D
Pick     q6eor   Commit F
Reword   a2duw   Commit E
```

```
# This is a combination of 2 commits.
# The first commit's message is:

Squashed Commit

[1] <rge/git-rebase-todo [+][gitrebase][Git(master)]    All
:wq
```

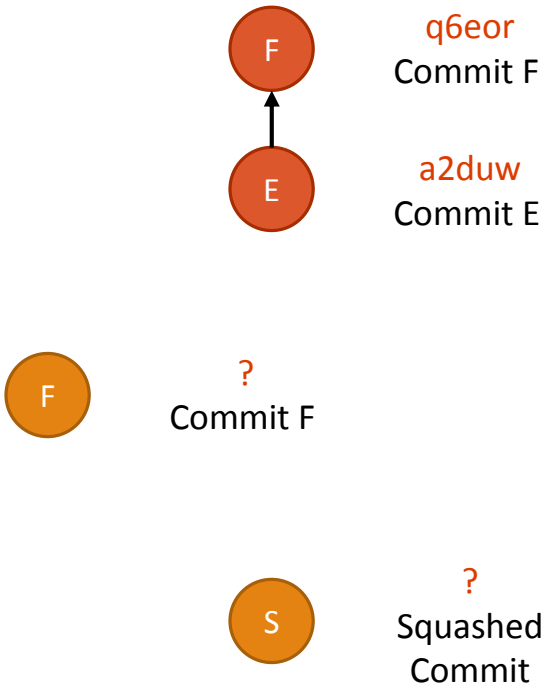


Rebasing

$ git rebase -i

```
Pick     d9e5q   Commit B
Squash   s7dyw   Commit C
Squash   8dyad   Commit D
Pick     q6eor   Commit F
Reword   a2duw   Commit E
```

```
[detached HEAD 056bfd5] Squashed Commit
 Date: Mon Mar 13 22:18:33 2017 +0200
 1 file changed, 1 insertion(+), 1
deletion(-)
Auto-merging File.txt
CONFLICT (content): Merge conflict in
File.txt
error: could not apply q6eor7... Commit F

$ git add File.txt

$ git rebase --continue
```



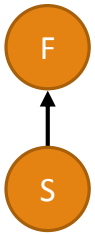
Rebasing

$ git rebase -i

```
Pick     d9e5q   Commit B
Squash   s7dyw   Commit C
Squash   8dyad   Commit D
Pick     q6eor   Commit F
Reword   a2duw   Commit E
```

```
Commit E

# Please enter the commit message for
your changes. Lines starting with # will
be ignored, an empty message aborts the
commit
# ...
# ...

[1] <rge/git-rebase-todo [+][gitrebase][Git(master)]    All
:wq
```

a2duw
Commit E



?
Commit F

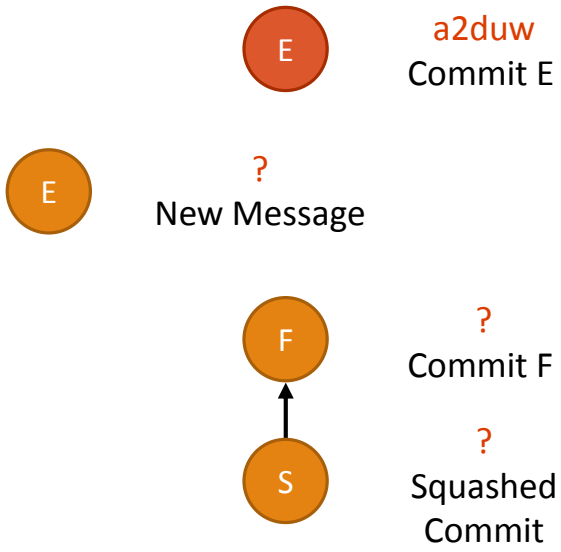
?
Squashed Commit

Rebasing

$ git rebase -i

```
Pick     d9e5q   Commit B
Squash   s7dyw   Commit C
Squash   8dyad   Commit D
Pick     q6eor   Commit F
Reword   a2duw   Commit E
```

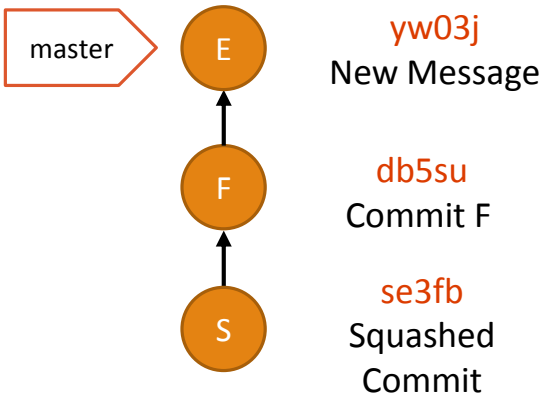
```
New Message

# Please enter the commit message for
your changes. Lines starting with # will
be ignored, an empty message aborts the
commit
# ...
# ...

[1] <rge/git-rebase-todo [+][gitrebase][Git(master)]    All
:wq
```



Rebasing

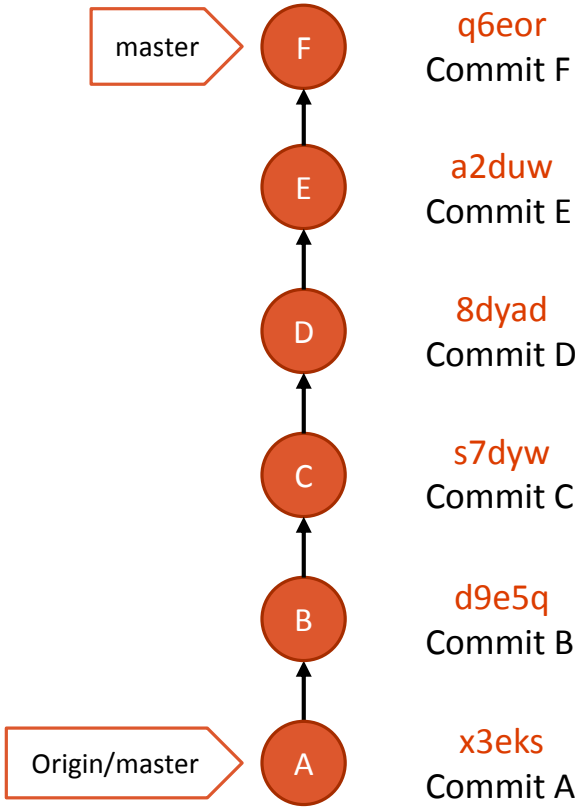
$ git rebase -i

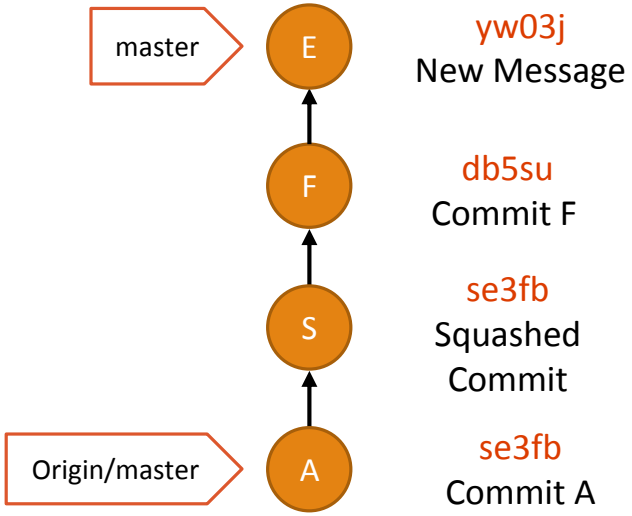
```
$ git rebase -i

[detached HEAD 056bfd5] Squashed Commit
 Date: Mon Mar 13 22:18:33 2017 +0200
 1 file changed, 1 insertion(+), 1
deletion(-)
Auto-merging File.txt
CONFLICT (content): Merge conflict in
File.txt
error: could not apply q6eor7... Commit F

$ git add File.txt

$ git rebase --continue

Successfully rebased and updated
refs/heads/master.
```



Rebasing

$ git rebase -i





Merging VS Rebasing

- If you aren't sure what you are doing, use merge (rebase can be dangerous)
- Use rebase to keep a clean story (rewrite history)
- Use merge to have a detailed history (although sometimes messy)
- Interactive rebase is great for cleaning changes before pushing
- Do not rebase pushed commits

Git is a Toolbox

- Knowing a command means understanding how it affects the 4 areas
- You can do the same things using different commands
- The same commands can do different things under different circumstances
- There are no better or worse in Git, just different ways of doing the same things

Git Workflows

Git workflow is a methodology that define:

- The Distribution Model

  How many repositories there are and who can access them?

- The Branching Models

  Which branches do you have and how you use them?

- The Constraints

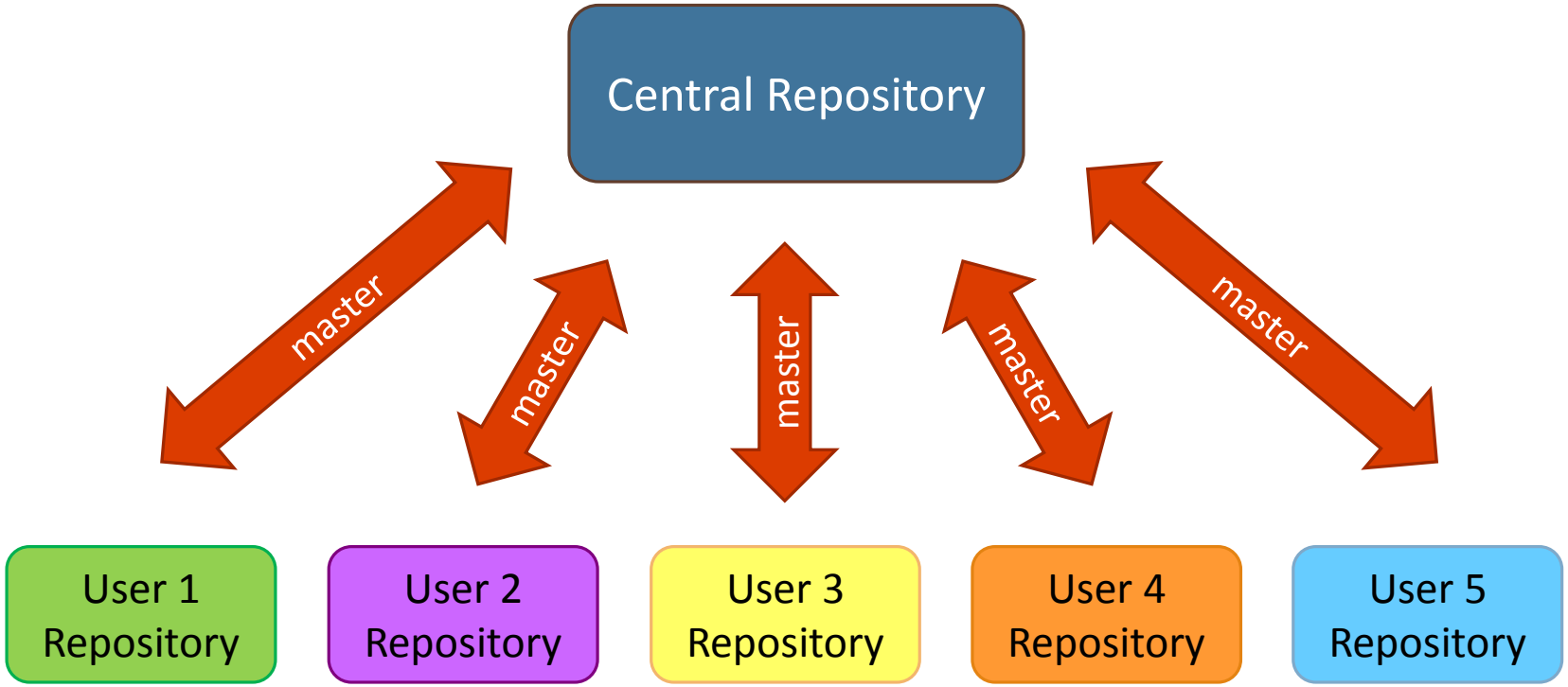
  What are the rules for working in the project?

Most Known Workflows

- Centralized Workflow
- Forking Workflow
- Feature Branch Workflow
- Environment Workflow
- GitFlow Workflow

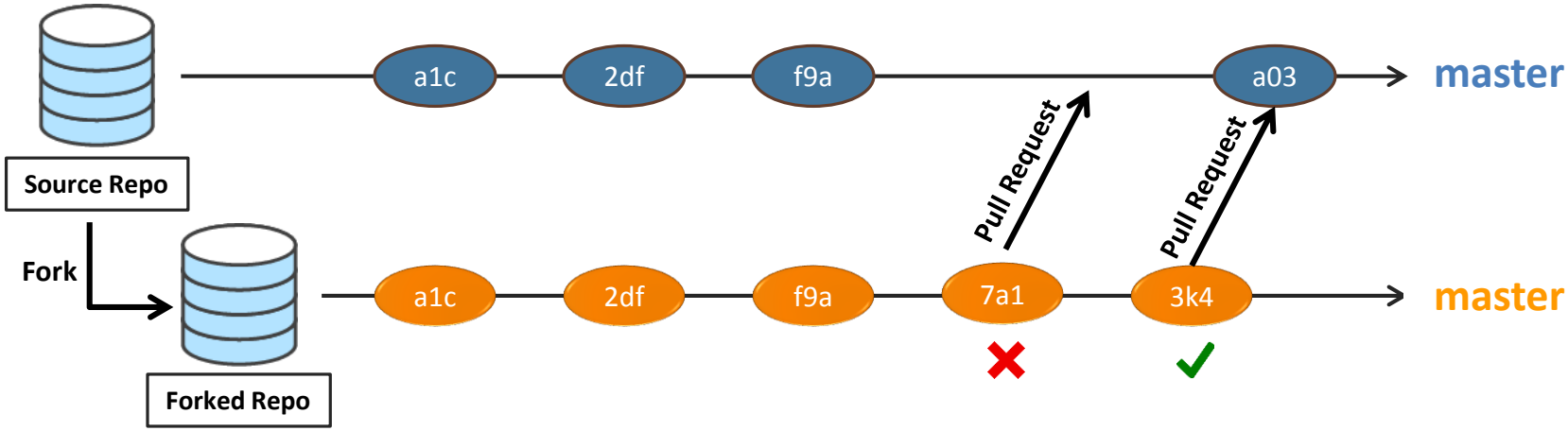
Most Known Workflows

Centralized Workflow



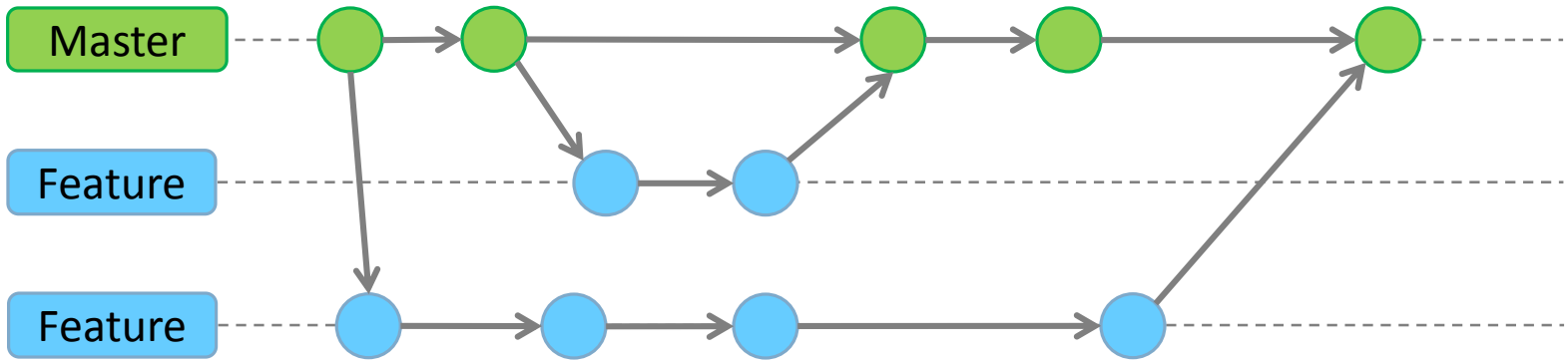
Most Known Workflows

Forking Workflow



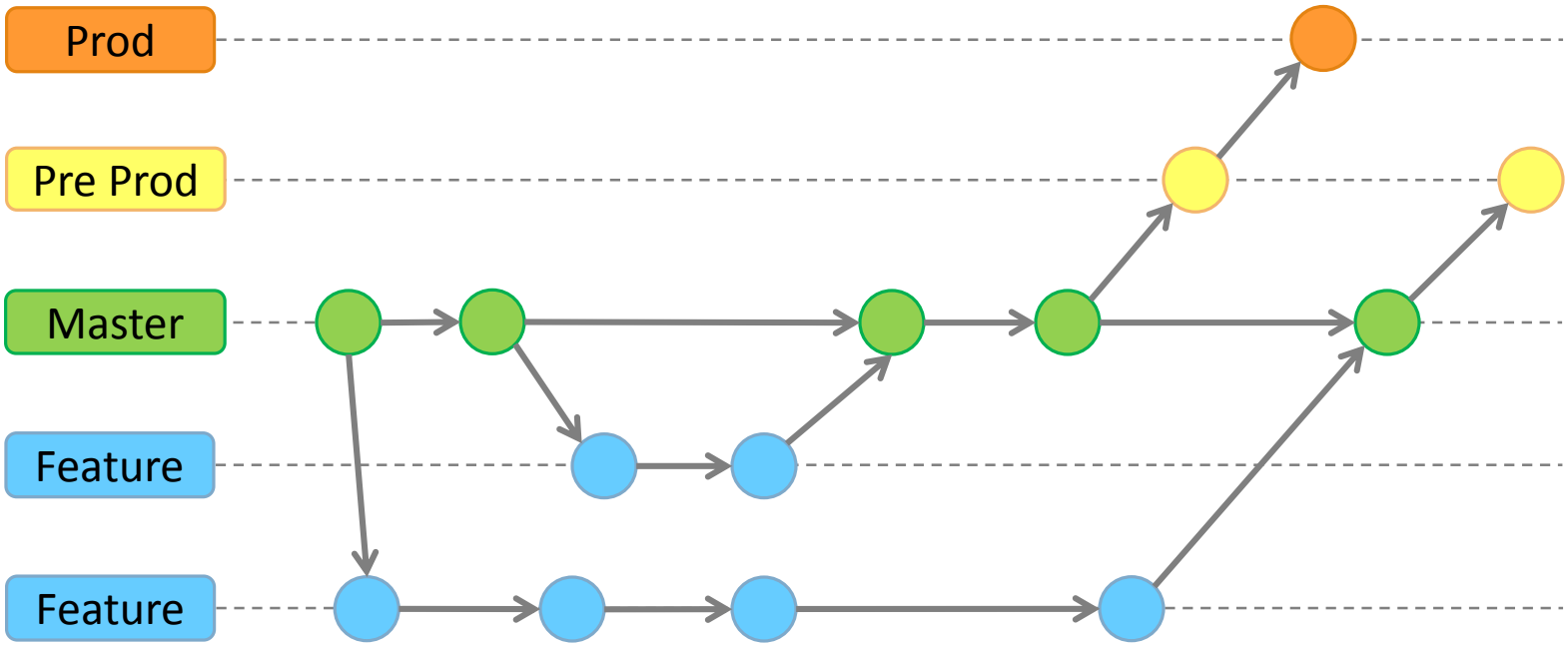
Most Known Workflows

Feature Branch Workflow



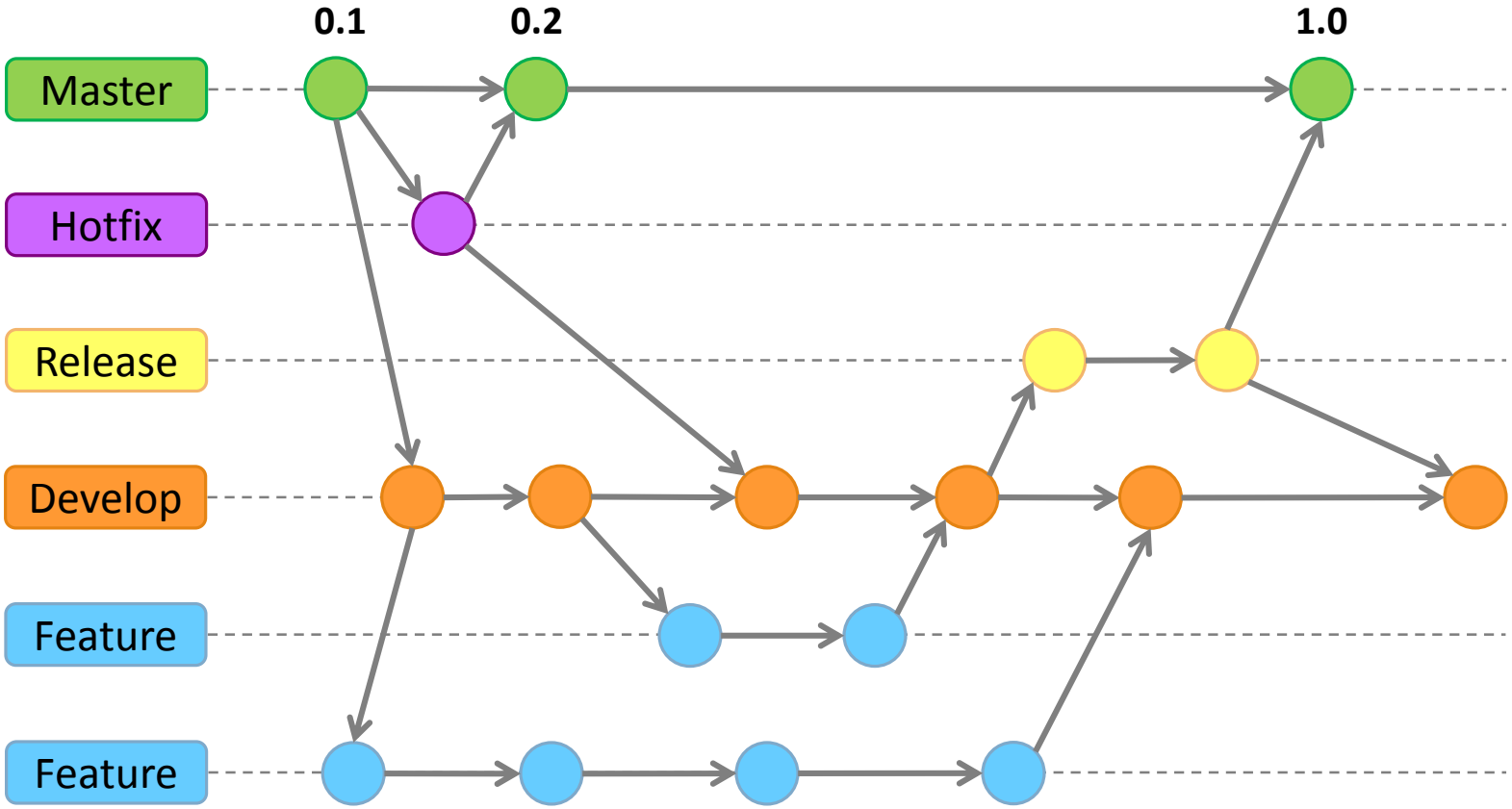
Most Known Workflows

Environment Workflow



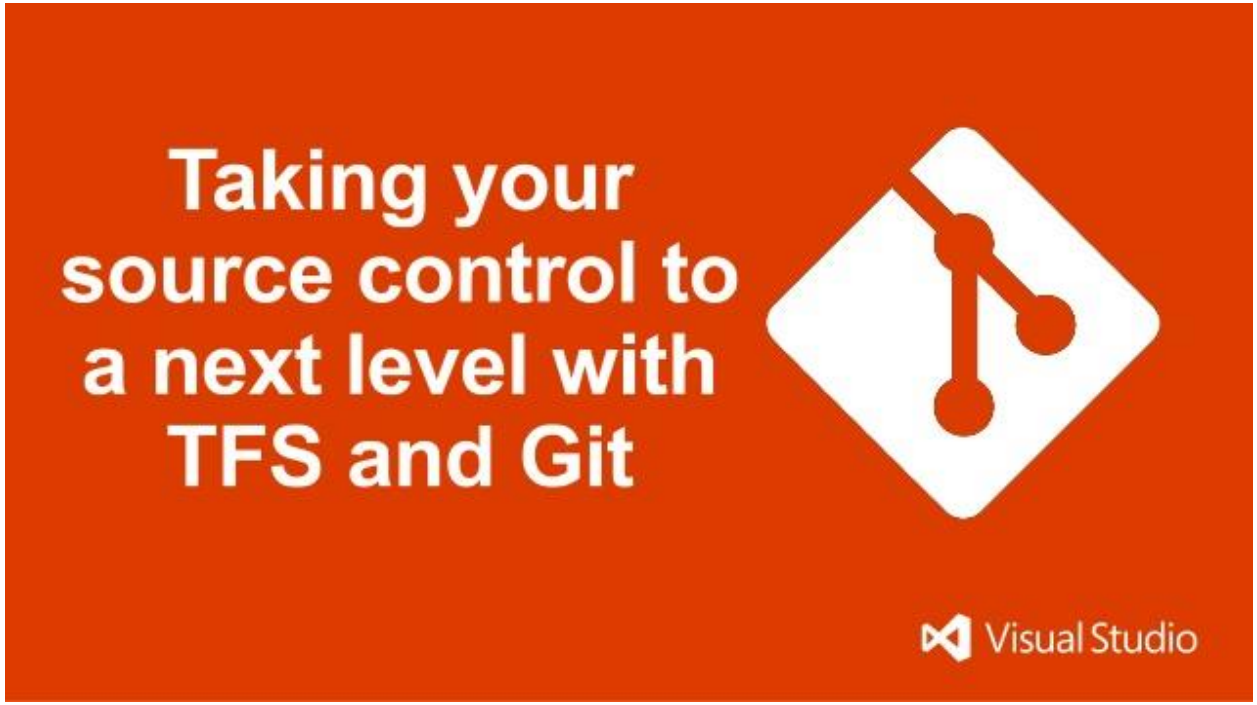
Most Known Workflows

GitFlow Workflow



Demo

- Feature Branch Workflow using TFS 2017



Advanced Git Overview

- git config
- git reflog
- git blame
- git hooks
- git gc
- git LFS
- git bisect
- git submodules

Thank you for coming!

I hope you enjoyed it…