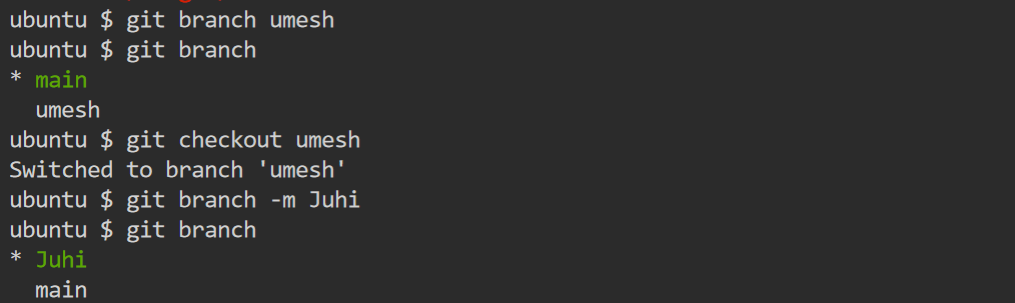
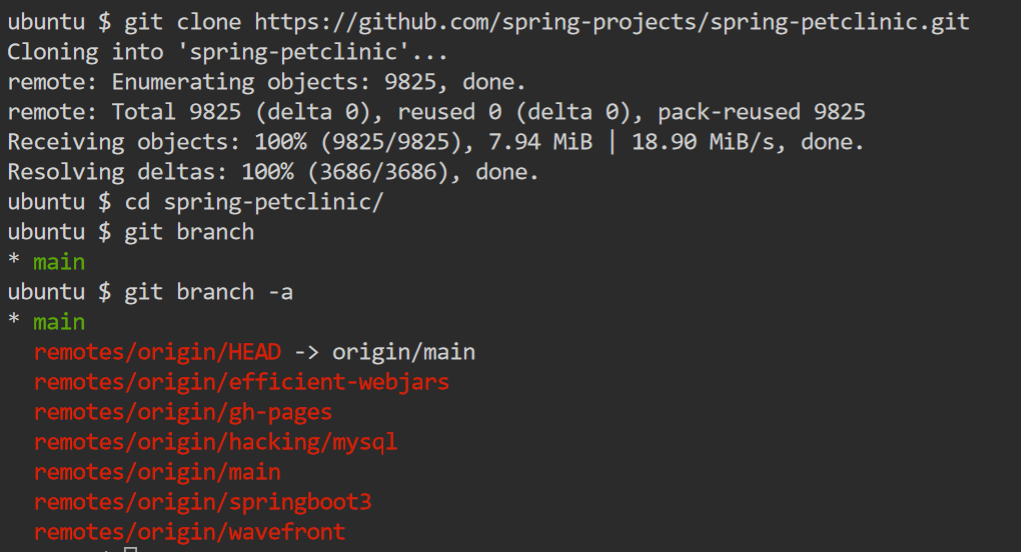
Branching: <https://www.atlassian.com/git/tutorials/using-branches>

Why we need to create branch? A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand-new working directory, staging area, and project history.

In Git, a branch is a new/separate version of the main repository.

* git branch or git branch --list
* git branch <branch>
* git branch -d <branch>
* git branch -D <branch>
* git branch -m <branch>
* git branch -a
* git branch crazy-experiment
* git remote add new-remote-repo https://bitbucket.com/user/repo.git
* git push <new-remote-repo> crazy-experiment
* git push origin --delete crazy-experiment





Merging: <https://www.atlassian.com/git/tutorials/using-branches/git-merge>

Why we need merging? To combine changes in master and child branch

**--skip --continue --abort**

A merged version reconciling the changes from all branches to be merged is committed, and your HEAD, index, and working tree are updated to it.

2-way merge

# Start a new feature

git checkout -b new-feature main

# Edit some files

git add <file>

git commit -m "Start a feature"

# Edit some files

git add <file>

git commit -m "Finish a feature"

# Merge in the new-feature branch

git checkout main

git merge new-feature

git branch -d new-feature

3-way merge

Start a new feature

git checkout -b new-feature main

# Edit some files

git add <file>

git commit -m "Start a feature"

# Edit some files

git add <file>

git commit -m "Finish a feature"

# Develop the main branch

git checkout main

# Edit some files

git add <file>

git commit -m "Make some super-stable changes to main"

# Merge in the new-feature branch

git merge new-feature

git branch -d new-feature

Resolving conflict

On branch main

Unmerged paths:

(use "git add/rm ..." as appropriate to mark resolution)

both modified: hello.py

here is some content not affected by the conflict

<<<<<<< main

this is conflicted text from main

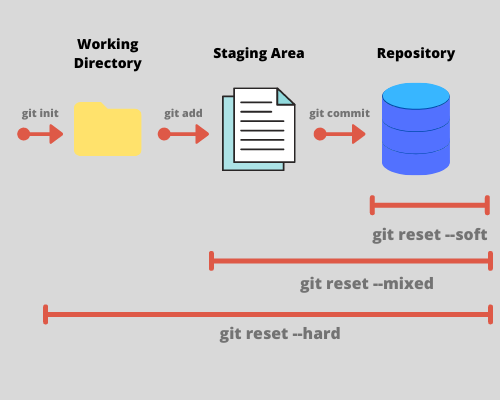
=======

this is conflicted text from feature branch

>>>>>>> feature branch;

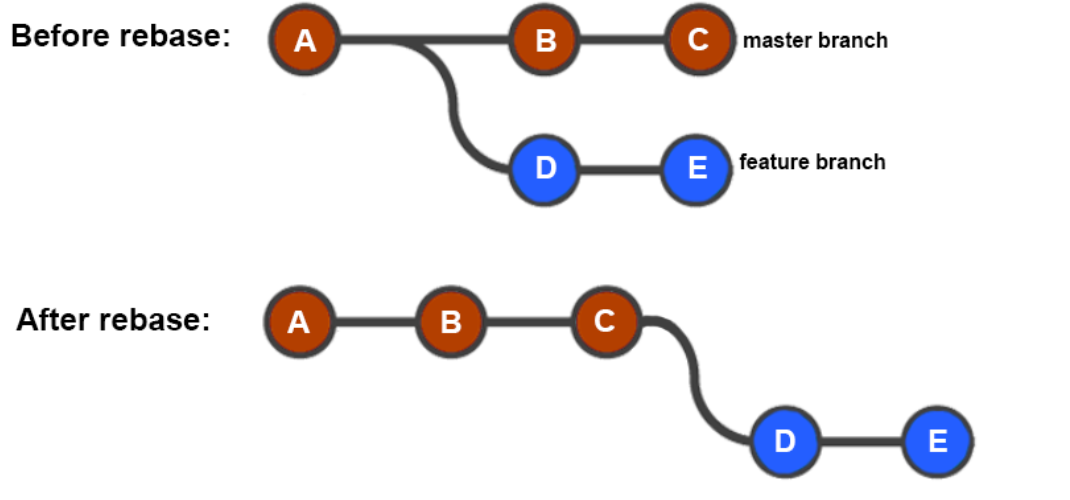
Reset: <https://www.atlassian.com/git/tutorials/undoing-changes/git-reset>

The git reset command is a complex and versatile tool for undoing changes. It has three primary forms of invocation. These forms correspond to command line arguments --soft, --mixed, --hard .



Rebase: <https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase>

Rebasing in Git is a process of integrating a series of commits on top of another base tip. It takes all the commits of a branch and appends them to the commits of a new branch.

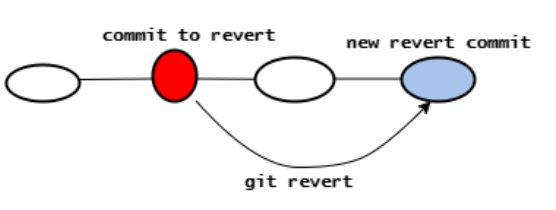


Cherry-pick: <https://www.atlassian.com/git/tutorials/cherry-pick>

Cherry-picking in git means choosing a commit from one branch and applying it to another branch. This is in contrast with other ways such as merge and rebases which normally apply many commits into another branch.

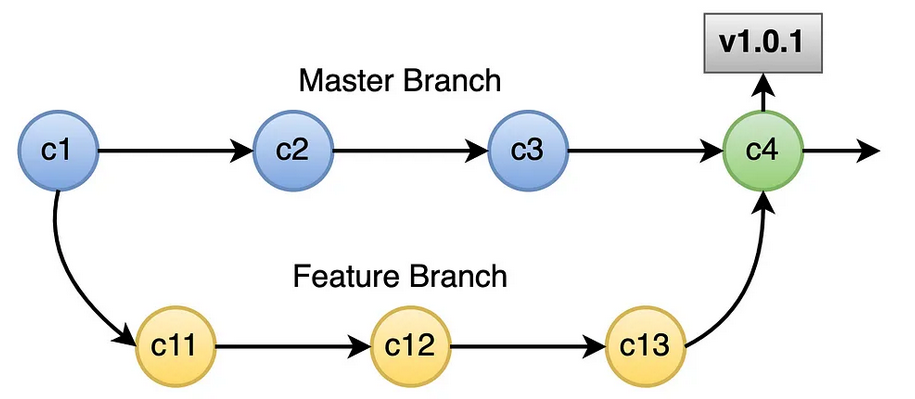
Revert: <https://www.atlassian.com/git/tutorials/undoing-changes/git-revert>

The git revert command is a forward-moving undo operation that offers a safe method of undoing changes.



Tag: <https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag>

A Git Tag is used to **label** and **mark** a **specific commit** in the git commit history. It is usually used to mark release points (eg. v1.0.1, v1.0.2, etc.)



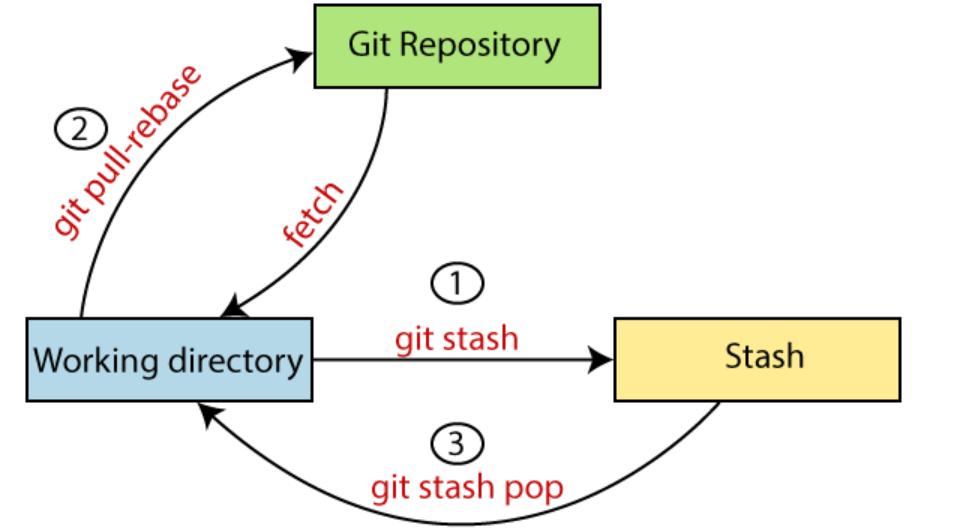
Fork: <https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow>

GIT Branching strategy: Trunk based Development

<https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development>

GIT stash: <https://www.atlassian.com/git/tutorials/saving-changes/git-stash>

Saving changes locally.



Casestudy:

* Rebase Branching strategy
* GIT revert [It’s creating additional commit]
* GIThub: <https://docs.github.com/en>
* Gerrit