Git Complete: The definitive, step-by-step guide to Git

Section 1: Introduction

- source control is a type of record-up that tracks changes
- repository contains the files
  - -> working directory is on your personal computer, git knows about them
    - -> .git-folder contains the repository.
    - -> in between is the staging area, here files are kept pre-commit before they become part of the version history
  - -> remote repository: final step
- master branch: default branch.

Section 2: Git installation

- ~~git-scm.com~~
- git-for-windows.github.io

Section 3: Git Quick Start

GitHub > new repository
> open terminal > mkdir Projects >
> 'cd Projects/'
> git config --global user.name "..."
user.email "..."
git config --global --list

Cloning a repository

GitHub > repository > code
> HTTPS
> git clone URL
> cd repos-name (ls for listing files)
> git status

Adding a file

echo "contents" >> file.txt
> cat file.txt to see contents

Stage a file

all

git add file.txt ✗ git add .

> git status

-> not yet committed

-> if not added, nothing for message

> git commit (-m "message")

first update before pushing: test track. -> from staging to local repository.

> git pull origin master

> git push origin master

copy of cloned branch

remote repo

Configure GitHub

find code application > copy address

> this PC > properties > advanced syst. set.

> Environment var. > syst. var. > Path >

new > paste

cat ~/.gitconfig

> git config --global

core.editor "pycharm64.exe

--multiInst --nosession"

Section 3: Basic git commands

Initialize new

git init fresh-project > cd fresh-project/ (working directory)

> ls -al (even see git repo)

Delete

rm -rf fresh-project/

Add git to existing project

mv old-folder-name new-name

cd web-project/ >
git init > ls -al > we can see the git folder
git add . > (add all to staging)
git commit -m "message"

rm -rf .git (only delete git comm.)

joining an existing project

GitHub > repo > fork > code > clone

git clone URL (git names folder after cloned repo)

Shortcut commit

git commit -a -m "Message"
add message

-> even within the same file we can have some changes that are staged and some that are not

More folders

mkdir -p folder1/folder2/...

Revert changes

git checkout -- file_name

-> clears out and undoes changes on file

Rename & move files

git mv old-name new-name

or mv old-name new-name
→ git does not see this as a rename but as a delete & new file
→ solve with git add -A

move by doing the rename again, this is easier

git mv file-name new-folder

if we manually rename, git sees it as two separate things
→ no git add -A since we do not want to track the system file
→ git add new-file
git add -u to see renaming

Delete files

-> file that is not yet being tracked (git ls-files)
rm file_name

-> file that is being tracked
git rm file_name
git commit -m "msg."

-> unstage delete
git reset HEAD file_name
-> delete from working dir.
git checkout -- file_name

-> can also do it not using git
rm file_name
git add -A

rm -rf folder_name
git add -A

Look at history

git help log

git log
-> string of char. is the unique ID

git log --abbrev-commit
-> shorter commit ID

git log --oneline --graph
--decorate

git log commitnn ... commitnn2

git log --since="3 days ago"

git log --file_name (only for certain file)

git log --follow --file_name
(also follows renames)

git show commitnn

Alias

-> create our commands

git config --global. → alias
→ not only this repository

alias.last
" command to alias without 'git' "

-> can use git last now
-> is now under
cat ~/.gitconfig

Ignore files

- .gitignore is text file with all files we need to ignore
  -> one expression per line
  e.g. MyFile.ext (one file.)
  *.ext (all extensions)
  my-folder/ (one folder)

-> git add .gitignore

Section 6 : Visual Merge/Diff Tool

-> P4 Merge : only Merge & Diff Tool

-> add Perforce folder to PATH
(this PC > properties > advanced > environment var > system var)

git config --global merge.tool p4merge

git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"
← where to look

git config --global mergetool.prompt false
← launch if need to resolve conflict

-> same 3 commands but replace with diff.tool and difftool

git config --global --list

Section 7: Comparisons

(1) `git diff HEAD`
→ diff working dir and last commit

`git difftool HEAD` does the same but visually

(2) `git diff`
→ diff on working dir and staging area
`git difftool`

(3) `git diff --staged HEAD` (HEAD ↑ last commit)
→ diff on staging and vs last commit

`git difftool --staged HEAD`

(4) `git diff -- file_name`

⑤ git log --oneline

git diff commit1 commit2
→ see changes between 2 commits

git diff HEAD HEAD^
→ compares head & head - 1

⑥ git diff master origin/master
local remote
→ difference commit & push

Session 8: Branching & merging

-> should experiment changes in feature or topic branches & integrate if stable

git branch -a (* ind. active)

create ↓ switch

git branch new_branch

git checkout new_branch

git branch -m old_name new_name

git branch -d branch-to_delete

git branch (local branches)

git checkout -b new_branch
-> create and switch branch

Merge in changes

git diff master branch1

git merge branch1 (if we can. already in the master)

-> fast forward merge: place all changes on master as if we never branched

git branch -d branch1 (delete br.)

git merge branch1 --no-ff
-> no fast forward, adds merge commit
-> git log --online
--graph
--decorate
preserves the branch past structure

merges after changes to master

git log --oneline --graph
--decorate --all
will show different commits
on branch and master

git merge branch1 -m "message"

merge conflicts

-> changes in different branches
on the same file

git diff branch1 master

git merge branch1 will try
to merge, but results in conflict

-> we are now in an in-between
stage
-> open conflicting file and
git will indicate differences
<<< ...
=== ...
>>> ...

git mergetool

-> PyMerge opens
-> decision on correct changes
-> save file

git commit -m "mess."

-> git saves .orig file with conflicts
-> can be added to ignore list

Section 9: Rebasing

- incorporate changes from master into feature branch
  -> goto feature branch

git rebase master
-> rebases all changes on new master

-> if rebase gives errors, we can abort it by following it up with git rebase --abort

-> or we can
git merge feature

Release incoming changes.

git fetch origin master
->only updates versions (i.e. my o dispenser in git folder)

-> want to keep commits ahead of changes on github without pushing changes
-> just integrate online changes

git pull --rebase origin master

Section 10: Stashing

-> not ready to commit a file, but have to work on another

git stash

-> deletes files from git staging area and undoes changes
-> after doing other work:

git stash apply

-> puts back into and redoes changes

git stash list shows all stashes in the stash

git stash drop deletes its contents

Stashing an untracked file

-> new file must be added to staging first or

git stash -u (include untr. files)

git stash pop applies stash and drops it

Multiple stashes

git stash save "message"
-> multiple times for each changed file

git stash list
-> last stash even index 0

git stash show stash@{nr} see changes

git stash apply stash@{nr}

git stash drop stash@{nr}
-> index of other stashes change

git stash clear deletes every thing

Stashing in a branch

-> move file to feature branch.

git stash -u

git stash branch branch_name
-> new branch + stash
applied & dropped

Section 11: Tagging

-> mark milestones

git tag tag-name

-> lightweight tag on current commit

git tag --list

-> can be used in other comm.

git show tag_name shows all related to that commit

git tag --delete tag-name

Annotated tag

git tag -a tag-name

annotated

-> editor is brought up for annotation

git commit --amend
-> amended commit msg.

git tag tag-name -m "msg"

git diff tag1 tag2

Tagging previous commit

git tag -a tag-name commitno.

Move tag

git tag -a tag-name -f commitno
(-f: force, commitno: correct id)

-> in github > tags on github we can view them after

git push origin tag-name

-> commits associated with tag are also available

git push origin master --tags
uploads all tags

git push origin :tag_name
-> push nothing to tag
-> no delete tag

Section 12: Bonus

git reset HEAD^1 to reset 1 commit

git reflog shows full history

3 resets
- mixed: reset what's in it
- hard: reset wd + staging area and move head
- soft: only point head

-> look at reflog for id of commit you want to go back to

-> git reset commit sha

git help cmd

cherry-pick: apply commit somewhere else

git cherry-pick commitsha