Git

Links

Overview

- Scott Chacon and Ben Straub: Pro Git book
- NDP Software: Online Interactive Cheat Sheet
- Linux Kernel: Git User Manual
- Git: giteveryday - A useful minimum set of commands for Everyday Git
- Atlassian: Gitflow Workflow

Merge

- Ted Felix: Git Conflict Resolution
- Skybert: Diffing and merging in Emacs

Rebase

- Bitbucket: Merging vs. Rebasing
- Sourcetree: Merge or Rebase?
- Stack Overflow: What's the difference between 'git merge' and 'git rebase'?
- Stack Overflow: Difference between git pull and git pull --rebase
- Stack Overflow: Git branch diverged after rebase

Submodules

- Stack Overflow: Git Submodule Workflow Advice

Hooks

- Stack Overflow: Prevent commits in master branch

Commands

Cloning

- clone only a single branch/tag (*Linux* kernel example)

```
git clone --depth 1 --single-branch --branch v5.10.52 \
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

```
linux-5.10.52
```

Status

- show status (long): `git status`
- show status (short): `git status -s`

Commit

- cherry pick a commit to index: `git cherry-pick -n -Xpatience <commit>`
- revert all commits, not pushed so far, up to `<last_good_commit>`, and keep all changes in worktree/index: `git reset --soft <last_good_commit>`
- revert/delete all commits after `<last_good_commit>` up to HEAD: `git reset --hard <last_good_commit>`; if pushed in the past then do in addition: `git push -f origin <branch>`

Merge/Rebase

- typical merge: `git merge --no-commit master`
- rebase a feature branch: `git checkout master;git pull;git checkout <branch>; git rebase master`

Some notes regarding *Rebase*:

1. In most cases rebasing does not produce any problem (it is best practice).
2. Interactive rebase (option `-i`) provides better control sometimes.
3. A rebase changes all commit hashes (Git rewrites the commit history) - so do not rebase commits that you've pushed!
4. Also regard the *Golden Rules* from Bitbucket.

- rebase a branch, w/o flatten it's merge commits: `git rebase -p`
- interactive edit/re-arrange/squash the last 3 commits (example): `git rebase -i HEAD~3`

Branches

- list remote branches: `git branch -r`
- show information about *origin*: `git remote show origin`
- create (and switch to) branch: `git checkout -b <branchname>`
- create branch and take over all working tree changes: `git stash branch <branchname>`
- push local branch to *origin* (create remote branch on demand): `git push origin <branchname>`
- push also the tags (this is not automatic!): `git push --tags`
- change remote tracking branch: `git branch --set-upstream-to=origin/<any> <branchname>`
- print name of current branch: `git rev-parse --abbrev-ref HEAD`; or with a *Bash* alias:

```
git_branch_name()
{
    local name="$(git rev-parse --abbrev-ref HEAD)"
    echo "$name"
}

alias gbn=git_branch_name
```

- print revision of branch base (gbb):[1] `git merge-base --fork-point master`

Log

- print summary of commits in branch: `git log --first-parent --pretty=oneline $(gbb)~..HEAD`; or: `git show-branch --current --sha1-name`
- show changed files in branch (local): `git diff --name-status $(gbb)~ HEAD --`
- show graphical history: `git log --graph --oneline --all`
- list last operations: `git reflog`

Tags

- list all tags, incl. comments: `git tag -n`
- create (annotated) tag: `git tag -a v0.5 -m “release 0.5”`

Diff

- create a (mbox) patchset, to be applied with `git am`:[2] `git format-patch -s -M --stat --summary --cover-letter -o mboxdir origin/devel..`
- create a patch, to be applied with `patch`:[3] `git diff [--cached] > patchfile`
- create a patch (using *Diff*, not *Git*), to be applied with `patch`: `diff -Naur file1 file2 > patchfile` (-N »treat absent files as empty, -r »recursive, -a »treat all files as text, -u »unified context)
- Ediff with *Emacs*: `git difftool --tool=ediff file`

Prefer `git format-patch` / `git am` against `git diff` / `git apply`, because of it's better conflict resolution capabilities.

Patch

- apply a (mbox) patchset, created with `git format-patch`: `git am --reject mboxdir/mboxfile`
- send a (mbox) patchset as e-mail:[4] `git send-email --no-signed-off-cc --to="user1 <user1@dummy.com>" --cc="user2 <user2@dummy.com>" mboxdir/mboxfile/`
- apply a `patchfile` with `patch`: `patch -p1 < patchfile`

Repo

- show tracked remote repos: `git remote -v`
- change *origin* URL: `git config remote.origin.url ssh://<user>@<server>:<port>/<repo>`

NOTES:

1. Proper working of some commands can be ensured only in conjunction with my *Git* configuration (see section Files below).
2. In particular i use Emacs for most development tasks, also in *VC* or *Magit* mode (see also the *Git* `difftool` script for Ediff).

Troubleshooting

- Have I lost my changes after rebasing?[5)]
- Especially on *MinGW* the following error message is very common (with transport protocol HTTPS): `SSL certificate problem: unable to get local issuer certificate`. This problem can be solved by suppressing the issuer certificate validation with:

```
git config --global http.sslVerify false
```

This contradicts the idea of certificates and might be a security issue.

Files

- `/etc/gitconfig`: system-wide *Git* configuration file
- `~/.gitignore`: ignored files (see `core.excludesfile` in file `~/.gitconfig` below)
- `~/.gitmessage.txt`: commit template (see `commit.template` in file `~/.gitconfig` below)
- `~/.gitconfig`: users global *Git* configuration:

```
# -*- mode: conf -*-
#
# Copyright (C) 2018 Ralf Hoppe <ralf@dfcgen.de>
#
[user]
    name = Ralf Hoppe
    email = ralf.hoppe@ieee.org

[core]
    editor = emacs
    excludesfile = ~/.gitignore
```

```
[pager]
    branch = false

[color]
    diff = auto
    status = auto
    branch = auto
    interactive = auto
    status = auto
    diff = auto

[credential]
    helper = cache

[commit]
    template = ~/.gitmessage.txt

[log]
    abbrevCommit = true

[sendemail]
#   smtpserver = ???
    from = Ralf Hoppe <ralf.hoppe@ieee.org>
    envelopesender = ralf.hoppe@ieee.org
    suppressfrom = true # do not add From: address to the cc: list
    confirm = auto # confirm before sending on automatic adding of
addresses
    thread = false # no threading by git-send-email (see git-format-
patch)
[diff]
    tool = ediff
    guitool = ediff

[difftool]
    prompt = false # do not prompt before launch

[difftool.ediff]
    cmd = ediff.sh $LOCAL $REMOTE

[merge]
    tool = ediff

[mergetool]
    keepBackup = false

[mergetool.ediff]
    cmd = ediff.sh $LOCAL $REMOTE $MERGED $BASE
    keepBackup = false
    trustExitCode = true
[pull]
```

```
    rebase = merges
[push]
    # "git push" without any refspec will push the current branch out
to
    # the same name at the remote repository only when it is set to
track
    # the branch with the same name over there
    default = simple

[format]
    signoff = true # add Signed-off-by: line to the commit message
    thread = shallow # do threading in git-format-patch
```

1)
After a rebase this command prints the rebase point, not the original fork point.
2)
Conforming to *Linux* kernel submit guideline.
3)
There is no need for option `--no-prefix` if applied with `git apply`.
4)
Send as shallowed thread according to configuration option *format.thread* in file ~/.gitconfig.
5)
If after a rebase (or twiddling with *refs*) it seems that all commits were lost then use `git reflog show` in conjunction with `git reset`.