**GIT Commands**

For advanced Git usage, I usually leverage the GUI of GitHub or BitBucket. But the **GUI way** may not solve some requirements beautifully.

I found several Git fundamental concepts. They are quite useful, but may not be well-known.

Check it out and let me know what you think!

Q: What is the **.gitkeep** file?

You may know.gitignore very well. But what is .gitkeep?

By default, Git doesn’t track empty directories. A file must exist within it. We can use the .gitkeep file to add empty directories into a git repo.

Q: What’s **git cherry-pick**?

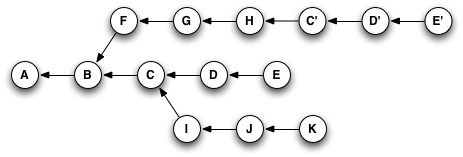
Pick changes from another branch, then apply them to the current branch.

And this usually happens, **before branch merge**.

$ git checkout $my\_branch

$ git cherry-pick $git\_revision

In the below figure, we cherry-pick C D E from branch2 to branch1.



Q: How to set up **two remote repo URLs** for one local git repo.

Sometimes I need to push before I can test, so I may push quite often. I don’t want to have too many tiny git pushes or too many push notifications for everyone. So what I usually do?

(Warning: this may not be a good practice for some projects, in terms of code security.)

Set up two remote repos for one local git repo. Keep pushing to one remote repo. **Once it’s fully tested, push to the official repo once**.

$ git clone git@github.com:devops/denny-repo.git

$ git config remote.myremote.url git@bitbucket.org:devops/denny-repo.git

$ git config remote.origin.url git@github.com:devops/denny-repo.git

$ cat .git/config

$ git push origin master # origin points to github

$ git push bitbucket master # myremote points to bitbucket

Q: What is **git stash**?

Stash local changes without git commit or git push. Switch to another branch, then come back later.

# Shelve and restore incomplete changes

# Temporarily stores all modified tracked files

$ git stash

# Restores the most recently stashed files

$ git stash pop

# Lists all stashed changesets

$ git stash list

# Discards the most recently stashed changeset

$ git stash drop

Q: What is **git rebase**?

git-rebase: Reapply commits on top of another base tip. If you don’t care about your git commit history, you can skip “git rebase,” but if you would prefer **a clean, linear history free of unnecessary merge commits**, you should reach for git rebase instead of git merge when integrating changes from another branch.

Q: What is **git squash**?

You may have several local git commits. Now run “git push,” and it will generate several git commit histories. To consolidate them as one, we can use the “git squash” technique.

# get local 3 latest commits

$ git rebase -i HEAD~3

# In editor, change "pick" to "squash". Keep the first one as "pick"

$ git rebase --continue

$ git push origin $branch\_name

Q: What is **git revert**?

A fun metaphor is to think of Git as timeline management utility. Commits are snapshots of a point in time or points of interest. Additionally, multiple timelines can be managed through the use of branches.

When "undoing" in Git, you are usually moving back in time, or to another timeline where mistakes didn't happen.

* [Undo local changes] When the changes haven’t been pushed yet

If I want to totally discard local changes, I will use git checkout. When the repo is small, I might even run “git clone,” then git checkout.

$ git checkout $branch\_name

* [Undo public changes] When the changes have been pushed yet

$ git log -n 5

$ git revert $git\_revision

$ git push origin $branch\_name

* How to undo a git pull?

| **COMMAND** | **SCOPE** | **COMMON USE CASES** |
| --- | --- | --- |
| git reset | Commit-level | throw away uncommited changes |
| git reset | File-level | Unstage a file |
| git checkout | Commit-level | Switch between branches or inspect old snapshots |
| git checkout | File-level | Discard changes in the working directory |
| git revert | Commit-level | Undo commits in a public branch |
| git revert | File-level | (N/A) |

Q: What is **git reflog**?

Git maintains a list of **checkpoints** which can accessed using reflog.

You can use reflog to **undo merges**, **recover lost commits or branches** and a lot more.

Cheat sheet of my frequent Git commands:

| **NAME** | **SUMMARY** |
| --- | --- |
| Show latest history with one line for each | git log -n 10 –oneline |
| Check git log by patterns | git log –grep=”$pattern” |
| Check git log by files | git log — foo.py bar.py |
| Change the last commit message | git commit –amend |
| Check git configuration for current repo | git config –list |
| Delete local branch | git branch -D $branch |
| Delete remote branch | git push origin –delete $branch |
| Delete local tag | git tag -d $tag |
| Delete remote tag | git push –delete origin $tag |