Git Command Reference

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals. This page familiarizes some of the useful git commands while dealing with RDK code base.

checkout

To switch to another branch in your local work environment.

```
$ git checkout BRANCH_NAME
```

Synopsis

```
git checkout [-q] [-f] [-m] [<branch>]

git checkout [-q] [-f] [-m] --detach [<branch>]

git checkout [-q] [-f] [-m] [--detach] <commit>

git checkout [-q] [-f] [-m] [[-b|-B|--orphan] <new_branch>] [<start_point>]

git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] [--] <pathspec>…

git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] --pathspec-from-file=<file> [--pathspec-file-nul]

git checkout (-p|--patch) [<tree-ish>] [--] [<pathspec>…]
```

Options

- **-q , --quiet** **:** Quiet, suppress feedback messages.
- **--progress** **:** Progress status is reported on the standard error stream by default when it is attached to a terminal, unless `--quiet` is specified.
- **-f , --force** **:**When switching branches, proceed even if the index or the working tree differs from `HEAD`. This is used to throw away local changes.
- **-b <new_branch> :** Create a new branch named `<new_branch>` and start it at `<start_point>`
- **-B <new_branch> :** Creates the branch `<new_branch>` and start it at `<start_point>`; if it already exists, then reset it to `<start_point>`. This is equivalent to running "git branch" with "-f".
- **-t , --track** **:** When creating a new branch, set up "upstream" configuration.
- **--no-track** **:** Do not set up "upstream" configuration, even if the `branch.autoSetupMerge` configuration variable is true.

Example

```
$ git checkout rdk-next
```

checks out the rdk-next branch.

add

To stage changes[file modifications and deletions].Accepts arguments for files or directories within the project directory.

```
$ git add
```

Synopsis

```
git add [--verbose | -v]      [--dry-run | -n]    [--force | -f]    [--interactive | -i]     [--patch | -p]     [--edit | -e]     [--[no-]all | --[no-]ignore-removal | [--update | -u]]    [--intent-to-add | -N]   [--refresh]  [--ignore-errors]  [--ignore-missing]     [–renormalize]        [--chmod=(+|-)x]    [--pathspec-from-file=<file> [--pathspec-file-nul]]  [--] [<pathspec>…]
```

Options

$ git add [options]

- **-n , --dry-run** :Don't actually add the file(s), just show if they exist and/or will be ignored.
- **-v , --verbose** :Be verbose.
- **-f , --force** :Allow adding otherwise ignored files.
- **-i , --interactive :** Add modified contents in the working tree interactively to the index. Optional path arguments may be supplied to limit operation to a subset of the working tree.
- **-p , --patch** : Interactively choose hunks of patch between the index and the work tree and add them to the index. This gives the user a chance to review the difference before adding modified contents to the index.
- **-e , --edit** :Open the diff vs. the index in an editor and let the user edit it. After the editor was closed, adjust the hunk headers and apply the patch to the index.
- **-u ,–update** :Update the index just where it already has an entry matching <pathspec>. This removes as well as modifies index entries to match the working tree, but adds no new files.

Example

```
$ git add file1.c file2.c
```

Adds the file1.c and file2.c available in the current folder directory.

commit

Consists of a snapshot of the directory structure and file contents for the entire project. Record changes to the repository.

```
$ git commit
```

Synopsis

```
git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [–amend] [--dry-run] [(-c | -C | --fixup | --squash) <commit>] [-F <file> | -m <msg>] [--reset-author] [--allow-empty] [--allow-empty-message] [--no-verify] [-e] [–author=]  [--date=<date>] [--cleanup=<mode>] [--[no-]status]  [-i | -o] [--pathspec-from-file=<file> [--pathspec-file-nul]]  [-S[<keyid>]] [--] [<pathspec>…]
```

Options

- **-a , --all** : Tell the command to automatically stage files that have been modified and deleted, but new files you have not told Git about are not affected.
- **--p , --patch** : Use the interactive patch selection interface to chose which changes to commit.
- **-C <commit> , --reuse-message=<commit> :** Take an existing commit object, and reuse the log message and the authorship information (including the timestamp) when creating the commit.
- **-c <commit> , --reedit-message=<commit> :** Like -C, but with `-c` the editor is invoked, so that the user can further edit the commit message.
- **--squash=<commit> :** Construct a commit message for use with `rebase --autosquash`.
- **--branch :** Show the branch and tracking info even in short-format.
- **-F <file> , --file=<file> :** Take the commit message from the given file. Use - to read the message from the standard input.

Example

```
$ vim file1.c // edit the file1.c
$ rm file2.c
$ git commit -a
```

The command git commit -a first looks at the working tree, notices that file1.c modified and removed file2.c, performs necessary git add and git rm.

Push

To upload local workspace to remote repository

```
$ git push
```

Synopsis

```
git push [--all | --mirror | --tags] [--follow-tags] [--atomic] [-n | --dry-run] [--receive-pack=<git-receive-pack>] [--repo=<repository>] [-f | --force] [-d | --delete] [--prune] [-v | --verbose]
[-u | --set-upstream] [-o <string> | --push-option=<string>] [--[no-]signed|--signed=(true|false|if-asked)] [--force-with-lease[=<refname>[:<expect>]] [--no-verify] [<repository> [<refspec>…]]
```

Example

```
$ git push origin HEAD:refs/for/master
```

This will push the changes to master branch available in the remote location

branch

To view a list of existing branches

```
$ git branch
```

Creates a new topic branch

```
$ git branch [branch]
```

Synopsis

```
git branch [--color[=<when>] | --no-color] [--show-current] [-v [--abbrev=<length> | --no-abbrev]] [--column [=<options>] | --no-column] [--sort=<key>] [(--merged | --no-merged) [<commit>]] [--contains [<commit]] [--no-contains [<commit>]] [--points-at <object>] [--format=<format>] [(-r | --remotes) | (-a | --all)] [--list] [<pattern>…]

git branch [--track | --no-track] [-f] <branchname> [<start-point>]

git branch (--set-upstream-to=<upstream> | -u <upstream>) [<branchname>]

git branch --unset-upstream [<branchname>]

git branch (-m | -M) [<oldbranch>] <newbranch>

git branch (-c | -C) [<oldbranch>] <newbranch>

git branch (-d | -D) [-r] <branchname>…

git branch --edit-description [<branchname>]
```

Options

- **-d , --delete** **:** Delete a branch.
- **-D** **:** Shortcut for **`--delete --force`.**
- **--create-reflog** **:** Create the branch's reflog.
- **-f , --force** **:** Reset <branchname> to <startpoint>, even if <branchname> exists already. Without **`-f`,** git branch refuses to change an existing branch.
- **-m , --move** **:**Move/rename a branch and the corresponding reflog.
- **-c ,--copy** **:**Copy a branch and the corresponding reflog.
- **-l , --list** **:**List branches. With optional `<pattern>...`, e.g. `git branch --list 'maint-*'`, list only the branches that match the pattern(s).
- **-a , --all** **:**List both remote-tracking branches and local branches. Combine with `--list` to match optional pattern(s).
- **--show-current** **:** print the name of the current branch. In detached HEAD state, nothing is printed.

Example

```
$ git branch development
```

`creating a new git branch called “development”`

```
$ git branch -d development
```

Deleted development branch.

diff

Show changes between commits, commit and working tree, etc

```
$ git diff
```

Synopsis

```
git diff [<options>] [<commit>] [--] [<path>…]
git diff [<options>] --cached [<commit>] [--] [<path>…]
git diff [<options>] <commit> <commit> [--] [<path>…]
git diff [<options>] <blob> <blob>
git diff [<options>] --no-index [--] <path> <path>
```

Options

- **-p,–patch** :Generate patch
- **-U <n> , --unified=<n> :**Generate diffs with <n> lines of context instead of the usual three. Implies `--patch`. Implies `-p`.
- **--output=<file>** **:**Output to a specific file instead of stdout.
- **--raw** **:**Generate the diff in raw format.
- **--minimal** :Spend extra time to make sure the smallest possible diff is produced.
- **--anchored=<text> :** Generate a diff using the "anchored diff" algorithm.

Example

```
$ git diff           - Changes in the working tree not yet staged for the next commit.


$ git diff --cached  - Changes between the index and the last commit; what would be committing on running "git
commit" without "-a" option.


$ git diff HEAD     - Changes in the working tree since last commit; what would be committing on running "git
commit -a"
```

log

Show commit logs. Shows the history of the current branch.

```
$ git log
```

Shows the commits that aren't pushed.

```
$ git log m/[codeline]..
```

Synopsis

```
$ git log [<options>] [<revision range>] [[--] <path>…]
```

Options

- **--follow** **:**Continue listing the history of a file beyond renames (works only for a single file).
- **--source** **:**Print out the ref name given on the command line by which each commit was reached.
- **--log-size** **:**Include a line “log size <number>” in the output for each commit, where <number> is the length of that commit’s message in bytes.
- **-L <start>,<end>:< file> , -L :<funcname>:<file> :** Trace the evolution of the line range given by "<start>,<end>" (or the function name regex <funcname>) within the <file>.
- **-<number>n <number>-max-count=<number>** **:** Limit the number of commits to output.
- **--skip=<number> :** Skip number commits before starting to show the commit output.

Example

```
$ git log --no-merges
```

Show the whole commit history, but skip any merges

rebase

Reapply commits on top of another base tip.

```
$ git rebase [options]
```

Synopsis

```
git rebase [-i | --interactive] [<options>] [--exec <cmd>] [--onto <newbase> | --keep-base] [<upstream>
[<branch>]]
git rebase [-i | --interactive] [<options>] [--exec <cmd>] [--onto <newbase>] --root [<branch>]
git rebase (--continue | --skip | --abort | --quit | --edit-todo | --show-current-patch)
```

Options

- **-i , --interactive :** Make a list of the commits which are about to be rebased. Let the user edit that list before rebasing.
- **--continue** : Restart the rebasing process after having resolved a merge conflict.
- **--abort** :Abort the rebase operation and reset HEAD to the original branch.
- **--quit** : Abort the rebase operation but HEAD is not reset back to the original branch. The index and working tree are also left unchanged as a result.
- **--onto <newbase> :** Starting point at which to create the new commits. If the --onto option is not specified, the starting point is <upstream>. May be any valid commit, and not just an existing branch name.
- **--skip** : Restart the rebasing process by skipping the current patch.
- **--edit-todo** : Edit the todo list during an interactive rebase.
- **-m ,–merge** :Use merging strategies to rebase. When the recursive (default) merge strategy is used, this allows rebase to be aware of renames on the upstream side.
- **--show-current-patch** :Show the current patch in an interactive rebase or when rebase is stopped because of conflicts. This is the equivalent of `git show REBASE_HEAD`.

Example

```
$ git rebase --interactive
```

To squash a series of commits into a single commit.

stash

Stash command saves the local modifications away and reverts the working directory to match the `HEAD` commit.

Synopsis

```
git stash list [<options>]
git stash show [<options>] [<stash>]
git stash drop [-q|--quiet] [<stash>]
git stash ( pop | apply ) [--index] [-q|--quiet] [<stash>]
git stash branch <branchname> [<stash>]
git stash [push [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet] [-u|--include-untracked] [-a|--all] [-m|--
message <message>] [--] [<pathspec>…]]
git stash clear
git stash create [<message>]
git stash store [-m|--message <message>] [-q|--quiet] <commit>
```

Options

- **list [<options>]** : List the stash entries that you currently have. Eachstash entryis listed with its name (e.g.`stash@{0}`is the latest entry,`stash@{1}`is the one before, etc.),

- **show [<options>] [<stash>] :** Show the changes recorded in the stash entry as a diff between the stashed contents and the commit back when the stash entry was first created.
- **drop [-q|--quiet] [<stash>] :** Remove a single stash entry from the list of stash entries. When no`<stash>`is given, it removes the latest one. i.e. `stash@{0}`, otherwise`<stash>`must be a valid stash log reference of the form`stash@{<revision>}`.
- **branch <branchname> [<stash>] :** Creates and checks out a new branch named`<branchname>`starting from the commit at which the`<stash>` was originally created, applies the changes recorded in`<stash>`to the new working tree and index. If that succeeds, and`<stash>`is a reference of the form`stash@{<revision>}`, it then drops the`<stash>`. When no`<stash>`is given, applies the latest one.
- **clear :** Removes all the stash entries
- **create :** Create a stash entry (which is a regular commit object) and return its object name, without storing it anywhere in the ref namespace.
- **store :** Store a given stash created via git stash create (which is a dangling merge commit) in the stash ref, updating the stash reflog.

Example

```
$ git pull
 ...
file file1.c not up-to-date, cannot merge.
$ git stash
$ git pull
$ git stash pop
```