Git Cheatsheet

GIT CHEAT SHEET

Git is an open-source distributed version control system that helps manage source code versions locally and collaborate with others by integrating with platforms like GitHub. This cheatsheet summarizes frequently used Git commands for quick reference.

STAGE & SNAPSHOT

Work with the snapshot history and the staging area.

- `git status`: Show the status of modified files in the working directory and staging area.
- `git add [file]`: Stage changes for the next commit. (`git add .` for all, `git add -p` for parts)
- `git commit -m "msg"`: Commit staged changes. (`-am` to stage and commit tracked files)
- `git commit`: Commit and open text editor for message.
- `git commit --amend`: Change the last commit message or add forgotten files.
- `git reset HEAD`: Unstage everything.
- `git diff`: Show unstaged changes.
- `git diff --staged`: Show staged changes.
- `git diff HEAD`: Show all staged and unstaged changes.

SETUP

User information and global configurations.

- `git config --global user.name [name]`: Set the username to be recorded in the commit history.
- `git config --global user.email [email]`: Set the email address to be associated with the commit history.
- `git config --global color.ui auto`: Automatically apply colors to command outputs for better readability.

SETUP & INIT

Repository initialization and cloning.

- `git init`: Initialize the current directory as a new Git repository.
- `git clone [url]`: Clone a remote repository from the specified URL to the local machine.

BRANCH & MERGE

Manage independent work flows and merge them.

- `git branch`: List branches. (`--sort=-committerdate` to sort by recent commit)
- `git branch [name]`: Create a new branch.
- `git checkout [name]` / `git switch [name]`: Switch branches.
- `git checkout -b [name]` / `git switch -c [name]`: Create and switch to a new branch.
- `git merge [name]`: Merge changes from the specified branch.
- `git merge --squash [name]`: Combine all branch changes into one commit on current branch.
- `git branch -d [name]`: Delete a branch. (`-D` for force delete)
- `git log`: Display commit history. (`--oneline` for one line, `--graph` for tree view)

INSTALLATION & GUIS

Platform-specific installation and GUI tools.

- GitHub for Windows: https://windows.github.com
- GitHub for Mac: https://mac.github.com
- Git for All Platforms (Linux, Solaris, etc.): http://git-scm.com

SHARE & UPDATE

Synchronize and update with remote repositories.

- `git remote add [alias] [url]`: Add a remote repository.
- `git fetch [alias]`: Retrieve branch info from remote. (`--all` for all remotes)
- `git pull`: Fetch and immediately merge. (`--rebase` to fetch and rebase)
- `git push`: Transmit commits to remote. (`-u origin [name]` for first push)
- `git push --force-with-lease`: Safer force push.
- `git push --tags`: Push all tags.

TRACKING PATH CHANGES

Manage file deletions and path changes.

- `git rm [file]`: Delete file and stage deletion.
- `git rm --cached [file]`: Stop tracking file without deleting it.
- `git mv [old] [new]`: Rename/Move file and stage change.
- `git log --stat -M`: Display logs with movement/rename info.

TEMPORARY COMMITS

Temporarily store work-in-progress changes.

- `git stash`: Temporarily store current modifications and clear the working directory.
- `git stash list`: List all stashed changes.
- `git stash pop`: Apply the most recently stashed changes and remove them from the list.
- `git stash drop`: Delete a specific stash entry.

REWRITE HISTORY

Modify branches and rewrite history.

- `git rebase [branch]`: Move current branch commits to the end of another.
- `git reset --hard [commit]`: Reset staging and working directory to a specific state.
- `git reset HEAD^`: Undo most recent commit (keep working directory).
- `git checkout [file]` / `git restore [file]`: Discard unstaged changes to a file.
- `git checkout [commit] [file]` / `git restore --source [commit] [file]`: Get file version from commit.
- `git clean`: Delete untracked files.

INSPECT & COMPARE

View history and check changes.

- `git log`: View commit history.
- `git log [file]`: Show every commit that modified a file.
- `git log -S [text]`: Find commits that added/removed specific text.
- `git blame [file]`: Show who last changed each line of a file.
- `git show [SHA]`: View detailed info of a commit. (`--remerge-diff` for merges)
- `git diff [commit1] [commit2]`: Compare two commits. (`--stat` for summary)

IGNORING PATTERNS

Specify files to be excluded from tracking.

- `git config --global core.excludesfile [path]`: Set a global ignore pattern file for all local repositories.
- `.gitignore`: Create in the project root to specify files or directories to be ignored by Git.

Advanced Git Workflow

Advanced Git Rebase

- `git rebase -i HEAD~3`: Interactively modify or squash the last 3 commits.
- `git rebase --onto main feature`: Move a specific branch to another base branch.
- `git rebase --continue`: Continue the rebase process after resolving conflicts.
- `git rebase --abort`: Abort the rebase and return to the original state.
- `git rebase --skip`: Skip the current commit and proceed with the rebase.

Cherry-pick and Advanced Merging

- `git cherry-pick <SHA>`: Apply changes from a specific commit to the current branch.
- `git cherry-pick <start>..<end>`: Apply a range of commits at once.
- `git cherry-pick --no-commit <SHA>`: Add changes to the staging area without creating a commit.
- `git merge --no-ff [branch]`: Force creation of a merge commit even in a Fast-forward relationship.
- `git merge --squash [branch]`: Combine all changes from the branch into a single commit for merging.

Submodule Management

- `git submodule add <url> <path>`: Add another repository as a subdirectory of the current project.
- `git submodule init`: Initialize submodules.
- `git submodule update`: Update submodule contents.

- `git submodule update --remote`: Update submodules to the latest commit on the remote repository.
- `git submodule foreach git pull origin main`: Run pull in all submodules simultaneously.

Git Hooks and Automation

- `pre-commit`: Hook executed just before a commit; often used for linting or tests.
- `post-commit`: Hook executed just after a commit; used for notifications, etc.
- `pre-push`: Hook executed just before a push; used for final validations.
- `commit-msg`: Verifies if the written commit message follows specific rules.

Advanced Branch Management

- `git branch -m [old] [new]`: Rename a branch.
- `git branch --set-upstream-to=origin/main [branch]`: Set the upstream branch for a local branch.
- `git branch --unset-upstream`: Unset the configured upstream branch.
- `git branch -d --force [name]`: Force delete a branch even if work is not completed.
- `git branch --merged`: List branches that have already been merged.
- `git branch --no-merged`: List branches that have not yet been merged.

Advanced Git Stash

- `git stash push -m "message"`: Stash changes with a descriptive message.
- `git stash list`: View all stashed entries.
- `git stash show stash@{0}`: View changes in a specific stash entry.
- `git stash apply stash@{0}`: Apply a stash without removing it from the list.
- `git stash pop stash@{0}`: Apply a stash and remove it from the list.
- `git stash branch [name] stash@{0}`: Create and switch to a new branch from a stashed state.

Bug Tracking with Git Bisect

- `git bisect start`: Start a binary search to find the point where a bug was introduced.
- `git bisect bad <commit>`: Mark a commit as having the bug.
- `git bisect good <commit>`: Mark a commit as bug-free.
- `git bisect run <script>`: Automatically run binary search using a script.
- `git bisect reset`: End search and return to the original branch.

Multi-branch Work with Git Worktree

- `git worktree add [path] [branch]`: Check-out another branch in a separate directory to work on them simultaneously.
- `git worktree list`: View all active working trees.
- `git worktree remove [path]`: Remove a previously added working tree.
- `git worktree prune`: Clean up invalid working tree information.

Recovery with Git Reflog

- `git reflog`: Detail the history of changes to HEAD to find accidentally deleted commits.
- `git reflog show [branch]`: View change history for a specific branch.
- `git reset --hard HEAD@{2}`: Force recovery to a specific point recorded in reflog.
- `git cherry-pick HEAD@{2}`: Select and recover a specific commit from history.

Advanced Git Config

- `git config --global core.autocrlf true`: Automatically convert line endings between OS differences.
- `git config --global core.safecrlf true`: Verify data loss during line ending conversion.
- `git config --global pull.rebase true`: Use rebase by default instead of merge during pull.
- `git config --global init.defaultBranch main`: Set the default branch name.
- `git config --global alias.co checkout`: Set shortcuts for frequently used commands.

Git LFS (Large File Storage)

- `git lfs install`: Initialize LFS for large file management.
- `git lfs track "*.psd"`: Specify files with certain extensions for LFS tracking.
- `git lfs track "*.zip"`: Configure ZIP files to be managed by LFS.
- `git lfs ls-files`: List files currently managed by LFS.
- `git lfs pull`: Fetch files from the LFS remote server.

Deployment Archives with Git Archive

- `git archive --format=zip --output=release.zip main`: Export work as a ZIP archive.
- `git archive --format=tar.gz --output=release.tar.gz main`: Export as a TAR.GZ archive.
- `git archive --format=zip --prefix=project/ main`: Create an archive including a specific prefix path.

Useful Advanced Command Combinations

```
# Summary and stats of the last 10 commits
git log --oneline --stat -10

# Track full history of a specific file even if renamed
git log --follow -- [filename]

# Compare commit counts between specific branches
git rev-list --count main
git rev-list --count [branch]

# View commit history during a specific period
git log --since="2023-01-01" --until="2023-12-31"

# View commits performed by a specific author
git log --author="Name"

# Search by commit message content
git log --grep="bug fix"

# View logs with file diffs (patch)
git log -p [filename]

# See the full history across branches in a graph format
git log --graph --oneline --all
```