# Git 치트시트

## GIT CHEAT SHEET

Git은 로컬 환경에서 소스 코드의 버전을 관리하고 GitHub 등과 연동하여 협업할 수 있게 돕는 오픈소스 분산 버전 관리 시스템입니다. 이 치트시트는 자주 사용하는 주요 Git 명령어를 빠르게 참고할 수 있도록 정리한 것입니다.

### STAGE & SNAPSHOT

스냅샷 기록 및 스테이징 영역 작업

- `git status`: 현재 작업 디렉토리와 스테이징 영역의 상태를 표시합니다.
- `git add [file]`: 변경 사항을 스테이징 영역에 추가합니다. (`.` 전체 추가, `-p` 부분 추가)
- `git commit -m "msg"`: 스테이징된 변경 사항을 커밋합니다. (`-am`은 추적 중인 파일 스테이징 및 커밋)
- `git commit`: 텍스트 편집기를 열어 커밋 메시지를 작성합니다.
- `git commit --amend`: 마지막 커밋 메시지를 수정하거나 누락된 파일을 추가합니다.
- `git reset HEAD`: 스테이징된 모든 변경 사항을 취소합니다.
- `git diff`: 스테이징되지 않은 변경 사항을 비교합니다.
- `git diff --staged`: 스테이징 영역의 변경 사항을 비교합니다.
- `git diff HEAD`: 스테이징 및 미스테이징된 모든 변경 사항을 확인합니다.

### SETUP

사용자 정보 및 전역 설정

- `git config --global user.name [이름]`: 커밋 이력에 기록될 사용자 이름을 설정합니다.
- `git config --global user.email [이메일]`: 커밋 이력에 연결될 이메일 주소를 설정합니다.
- `git config --global color.ui auto`: 명령어 실행 결과에 가독성을 높여주는 색상을 자동으로 적용합니다.

### SETUP & INIT

저장소 초기화 및 복제

- `git init`: 현재 디렉토리를 새로운 Git 저장소로 초기화합니다.
- `git clone [url]`: 지정한 URL의 원격 저장소를 로컬로 복제해 옵니다.

### BRANCH & MERGE

독립적인 작업 흐름 관리 및 병합

- `git branch`: 브랜치 목록을 확인합니다. (`--sort=-committerdate` 최근 커밋 순 정렬)
- `git branch [이름]`: 새로운 브랜치를 생성합니다.
- `git checkout [이름]` / `git switch [이름]`: 브랜치를 전환합니다.
- `git checkout -b [이름]` / `git switch -c [이름]`: 브랜치 생성 및 전환을 동시에 수행합니다.
- `git merge [이름]`: 지정한 브랜치를 현재 브랜치에 병합합니다.
- `git merge --squash [이름]`: 해당 브랜치의 모든 커밋을 하나의 커밋으로 합쳐서 병합합니다.
- `git branch -d [이름]`: 브랜치를 삭제합니다. (`-D` 강제 삭제)
- `git log`: 커밋 이력을 확인합니다. (`--oneline` 한 줄 요약, `--graph` 트리 뷰)

### INSTALLATION & GUIS

플랫폼별 설치 및 GUI 도구

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

### SHARE & UPDATE

원격 저장소 동기화 및 갱신

- `git remote add [별칭] [url]`: 원격 저장소를 추가합니다.
- `git fetch [별칭]`: 원격 저장소 정보를 가져옵니다. (`--all` 모든 원격지 정보)
- `git pull`: 원격 내용을 가져와 병합합니다. (`--rebase` 리베이스 방식으로 가져오기)
- `git push`: 로컬 커밋을 원격에 전송합니다. (`-u origin [이름]` 첫 푸시 시 연결 설정)
- `git push --force-with-lease`: 안전한 강제 푸시를 수행합니다.
- `git push --tags`: 모든 태그를 푸시합니다.

### TRACKING PATH CHANGES

파일 삭제 및 경로 변경 관리

- `git rm [file]`: 파일을 삭제하고 스테이징 영역에 기록합니다.
- `git rm --cached [file]`: 실제 파일은 유지하고 Git 추적만 중단합니다.
- `git mv [기존] [신규]`: 파일 이동/이름 변경을 수행하고 스테이징 영역에 기록합니다.
- `git log --stat -M`: 파일 이동/변경 이력을 포함한 로그를 확인합니다.

### TEMPORARY COMMITS

작업 중인 변경 사항 임시 저장

- `git stash`: 현재의 수정 사항을 임시로 저장하고 작업 디렉토리를 깨끗하게 비웁니다.
- `git stash list`: 임시 저장된 변경 사항의 목록을 확인합니다.
- `git stash pop`: 가장 최근에 저장한 변경 사항을 다시 적용하고 목록에서 제거합니다.
- `git stash drop`: 특정 임시 저장 항목을 삭제합니다.

### REWRITE HISTORY

브랜치 수정 및 이력 재작성

- `git rebase [브랜치]`: 현재 브랜치의 커밋들을 지정한 브랜치 끝으로 옮깁니다.
- `git reset --hard [커밋]`: 작업 디렉토리와 스테이징 영역을 특정 커밋 상태로 강제 복구합니다.
- `git reset HEAD^`: 최근 커밋을 취소합니다 (작업 내역은 유지).
- `git checkout [파일]` / `git restore [파일]`: 파일의 스테이징되지 않은 변경 사항을 삭제합니다.
- `git checkout [커밋] [파일]` / `git restore --source [커밋] [파일]`: 특정 커밋의 파일 버전을 가져옵니다.
- `git clean`: 추적되지 않는 파일들을 삭제합니다.

### INSPECT & COMPARE

이력 조회 및 변경 사항 확인

- `git log`: 커밋 기록을 조회합니다.
- `git log [파일]`: 특정 파일을 수정한 모든 커밋을 확인합니다.
- `git log -S [텍스트]`: 특정 텍스트가 추가/제거된 커밋을 검색합니다.
- `git blame [파일]`: 파일의 각 라인을 마지막으로 수정한 사람을 확인합니다.
- `git show [SHA]`: 특정 커밋의 상세 정보를 확인합니다. (`--remerge-diff` 병합 커밋 확인)
- `git diff [커밋1] [커밋2]`: 두 커밋 간의 차이를 비교합니다. (`--stat` 통계 요약)

### IGNORING PATTERNS

추적 제외 설정

- `git config --global core.excludesfile [경로]`: 모든 로컬 저장소에 적용할 전역 무시 패턴 파일을 설정합니다.
- `.gitignore`: 프로젝트 루트에 생성하여 특정 파일이나 디렉토리가 Git 추적 대상에서 제외되도록 지정합니다.

## 고급 Git 워크플로우

#### Git Rebase 심화

- `git rebase -i HEAD~3`: 최근 3개의 커밋을 대화형으로 수정하거나 합칩니다.
- `git rebase --onto main feature`: 특정 브랜치를 다른 베이스 브랜치로 옮깁니다.
- `git rebase --continue`: 충돌을 해결한 후 리베이스 작업을 계속합니다.
- `git rebase --abort`: 리베이스 작업을 중단하고 원래 상태로 되돌립니다.
- `git rebase --skip`: 현재 커밋을 건너뛰고 리베이스를 진행합니다.

#### Cherry-pick과 고급 병합

- `git cherry-pick <SHA>`: 특정 커밋의 변경 사항만 현재 브랜치에 적용합니다.
- `git cherry-pick <start>..<end>`: 특정 범위의 커밋들을 한꺼번에 적용합니다.
- `git cherry-pick --no-commit <SHA>`: 커밋을 생성하지 않고 변경 사항만 스테이징 영역에 추가합니다.
- `git merge --no-ff [브랜치]`: Fast-forward 관계라도 강제로 병합 커밋을 생성합니다.
- `git merge --squash [브랜치]`: 해당 브랜치의 모든 변경 사항을 하나의 커밋으로 합쳐서 병합합니다.

#### 서브모듈(Submodule) 관리

- `git submodule add <url> <경로>`: 다른 저장소를 현재 프로젝트의 하위 디렉토리로 추가합니다.
- `git submodule init`: 서브모듈을 초기화합니다.
- `git submodule update`: 서브모듈의 내용을 업데이트합니다.
- `git submodule update --remote`: 원격 저장소의 최신 커밋으로 서브모듈을 갱신합니다.
- `git submodule foreach git pull origin main`: 모든 서브모듈에서 일괄적으로 pull을 실행합니다.

#### Git Hooks와 자동화

- `pre-commit`: 커밋 직전에 실행되는 훅으로, 코드 스타일이나 테스트 검증에 자주 쓰입니다.

- `post-commit`: 커밋 직후에 실행되는 훅으로, 알림 발송 등에 활용됩니다.
- `pre-push`: 푸시 직전에 실행되는 훅으로, 최종 검증 작업에 사용됩니다.
- `commit-msg`: 작성한 커밋 메시지가 특정 규칙을 따르는지 검사합니다.

### 고급 브랜치 관리

- `git branch -m [기존] [신규]`: 브랜치 이름을 변경합니다.
- `git branch --set-upstream-to=origin/main [브랜치]`: 로컬 브랜치의 업스트림 브랜치를 설정합니다.
- `git branch --unset-upstream`: 설정된 업스트림 브랜치를 해제합니다.
- `git branch -d --force [이름]`: 작업이 완료되지 않았더라도 브랜치를 강제로 삭제합니다.
- `git branch --merged`: 이미 병합된 브랜치 목록을 확인합니다.
- `git branch --no-merged`: 아직 병합되지 않은 브랜치 목록을 확인합니다.

### Git Stash 고급 활용

- `git stash push -m "메시지"`: 설명과 함께 변경사항을 임시 저장합니다.
- `git stash list`: 저장된 모든 스태시 목록을 확인합니다.
- `git stash show stash@{0}`: 특정 스태시 항목의 변경 내용을 확인합니다.
- `git stash apply stash@{0}`: 스태시를 적용하되 목록에서 삭제하지는 않습니다.
- `git stash pop stash@{0}`: 스태시를 적용하고 목록에서 삭제합니다.
- `git stash branch [이름] stash@{0}`: 임시 저장된 상태에서 새로운 브랜치를 생성하여 이동합니다.

### Git Bisect로 버그 추적

- `git bisect start`: 버그가 발생한 지점을 찾는 이분 탐색을 시작합니다.
- `git bisect bad <커밋>`: 버그가 존재하는 커밋임을 표시합니다.
- `git bisect good <커밋>`: 버그가 없는 정상 커밋임을 표시합니다.
- `git bisect run <스크립트>`: 스크립트를 이용해 자동으로 이분 탐색을 실행합니다.
- `git bisect reset`: 탐색을 종료하고 원래 브랜치로 돌아갑니다.

### Git Worktree로 다중 브랜치 동시 작업

- `git worktree add [경로] [브랜치]`: 별도 디렉토리에 다른 브랜치를 체크아웃하여 동시에 작업합니다.
- `git worktree list`: 활성화된 모든 작업 트리 목록을 확인합니다.
- `git worktree remove [경로]`: 추가했던 작업 트리를 제거합니다.
- `git worktree prune`: 유효하지 않은 작업 트리 정보를 정리합니다.

### Git Reflog로 복구

- `git reflog`: HEAD의 변경 이력을 상세히 확인하여 실수로 삭제한 커밋 등을 찾습니다.
- `git reflog show [브랜치]`: 특정 브랜치의 변경 이력을 확인합니다.
- `git reset --hard HEAD@{2}`: reflog에 기록된 특정 시점으로 강제 복구합니다.
- `git cherry-pick HEAD@{2}`: 특정 시점의 커밋만 선택하여 복구합니다.

### Git Config 고급 설정

- `git config --global core.autocrlf true`: OS별 줄바꿈 차이를 자동으로 변환합니다.
- `git config --global core.safecrlf true`: 줄바꿈 변환 중 데이터 손실 여부를 검사합니다.
- `git config --global pull.rebase true`: pull 시 병합 대신 기본적으로 리베이스를 사용합니다.
- `git config --global init.defaultBranch main`: 기본 브랜치 이름을 설정합니다.
- `git config --global alias.co checkout`: 자주 쓰는 명령어의 단축어를 설정합니다.

### Git LFS (Large File Storage)

- `git lfs install`: 대용량 파일 관리를 위한 LFS를 초기화합니다.
- `git lfs track "*.psd"`: 특정 확장자 파일을 LFS 추적 대상으로 지정합니다.
- `git lfs track "*.zip"`: 압축 파일을 LFS로 관리하도록 설정합니다.
- `git lfs ls-files`: LFS로 관리 중인 파일 목록을 확인합니다.
- `git lfs pull`: LFS 원격 서버에서 파일을 가져옵니다.

### Git Archive로 배포용 아카이브 생성

- `git archive --format=zip --output=release.zip main`: 작업 내용을 ZIP 형식으로 묶어 내보냅니다.
- `git archive --format=tar.gz --output=release.tar.gz main`: TAR.GZ 형식으로 내보냅니다.
- `git archive --format=zip --prefix=project/ main`: 특정 접두사 경로를 포함하여 아카이브를 만듭니다.

### 유용한 고급 명령어 조합

```
# 최근 10개 커밋의 요약 정보와 통계 확인
git log --oneline --stat -10

# 파일 이름이 변경되었더라도 특정 파일의 전체 이력을 추적
git log --follow -- [파일명]

# 특정 브랜치 간의 커밋 개수 비교
git rev-list --count main
git rev-list --count [브랜치]

# 특정 기간 동안의 커밋 내역 조회
git log --since="2023-01-01" --until="2023-12-31"

# 특정 작성자가 수행한 커밋 조회
git log --author="Name"

# 커밋 메시지 내용으로 검색
git log --grep="bug fix"

# 파일 변경 내용(patch)과 함께 로그 확인
git log -p [파일명]

# 브랜치 전체의 히스토리를 그래프 형태로 한눈에 보기
git log --graph --oneline --all
```