Git Cheatsheet — Minimal & Professional

Concise, practical reference for daily Git usage (PowerShell / Bash variants shown)

Prepared for: Dean
Style: neutral, high-contrast (black/white), uniform formatting

Table of Contents

1. Basic Commands (Local)

The following commands are the essentials you will use daily. For each item: PowerShell (Windows) and Bash (Linux/macOS) variants are shown where they differ.

Check Git version

PowerShell / Bash:
```
git --version
```
Displays the installed Git version.

Initial configuration

PowerShell / Bash:
```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```
Sets your user name and email for commits.

Create a new repository

PowerShell / Bash:
```
git init
```
Initializes a new local Git repository in the current directory.

Clone a repository

PowerShell / Bash:
```
git clone https://.../.git
```
Copies a remote repository locally.

Check status

PowerShell / Bash:
```
git status
```
Displays unstaged/staged changes and current branch.

Stage a file

PowerShell / Bash:
```
git add <file>
```
Adds a file to the staging area for the next commit.

Stage all changes

PowerShell / Bash:
```
git add -A
```
Stages new, modified and deleted files.

Create a commit

PowerShell / Bash:
```
git commit -m "Message"
```
Records a snapshot with a commit message.

Amend the last commit

PowerShell / Bash:
```
git commit --amend
```
Modifies the last commit (use with caution if already pushed).

List branches

PowerShell / Bash:
```
git branch
```
Lists local branches.

Create and switch to a branch

PowerShell / Bash:
```
git switch -c feature-x
```
Creates a new branch and switches to it.

Switch branch (legacy)

PowerShell / Bash:
```
git checkout <branch>
```
Older command for switching branches; still supported.

Merge a branch

PowerShell / Bash:
```
git merge feature-x
```
Merges changes from 'feature-x' into the current branch.

Compact log

PowerShell / Bash:
```
git log --oneline --graph --decorate
```
Shows a compact, graphical history.

Discard changes (working copy)

PowerShell / Bash:
```
git restore <file>
```
Reverts a file to the last committed state.

Remove from staging

PowerShell / Bash:
```
git restore --staged <file>
```
Removes a file from the staging area.

Soft reset (undo commit, keep changes)

PowerShell / Bash:
```
git reset --soft HEAD~1
```
Resets the last commit but keeps changes in the working tree.

Hard reset (dangerous)

PowerShell / Bash:
```
git reset --hard HEAD~1
```
Discards commits and changes irreversibly (unless you have backups).

Stash changes (temporarily save)

PowerShell / Bash:
```
git stash
```

Temporarily stores uncommitted changes.

Apply and remove last stash

PowerShell / Bash:

```
 git stash pop
```

Reapplies the last stash and removes it from the stash list.

2. Remote Repositories & GitHub

Remote repositories let you share code and keep backups in the cloud. Use SSH for convenience and security; HTTPS works too.

Add a new remote (SSH)

```
git remote add origin git@github.com:USER/REPO.git
```
Links local repository to a GitHub repository via SSH.

Add a new remote (HTTPS)

```
git remote add origin https://github.com/USER/REPO.git
```
Links local repository using HTTPS.

Show remotes and URLs

```
git remote -v
```
Displays configured remotes and their URLs.

Change remote URL (to SSH)

```
git remote set-url origin git@github.com:USER/REPO.git
```
Switches an existing remote URL to SSH.

Change remote URL (to HTTPS)

```
git remote set-url origin https://github.com/USER/REPO.git
```
Switches an existing remote URL to HTTPS.

Push (first time, set upstream)

```
git push -u origin main
```
Pushes the branch and sets the upstream tracking branch.

Push (regular)

```
git push
```
Uploads local commits to the remote repository.

Pull (fetch + merge)

```
git pull
```
Fetches from remote and merges into current branch.

Fetch only (no merge)

```
git fetch
```
Downloads objects and refs; merge manually if needed.

Push tags

```
git push --tags
```
Pushes local tags to the remote.

Remove a remote

```
git remote remove origin
```
Removes the configured remote link.

Clone (SSH)

```
git clone git@github.com:USER/REPO.git
```
Copies a remote repository locally via SSH.

Clone (HTTPS)

```
git clone https://github.com/USER/REPO.git
```
Copies a remote repository locally via HTTPS.

Quick guide — create a GitHub repo and push (SSH):

1. On GitHub: Create a new repository (name: e.g. 'pytasks'). Do not initialize with a README if you already have a local repo.
2. Locally: git remote add origin git@github.com:YOUR_USER/pytasks.git
3. Locally: git branch -M main
4. Locally: git push -u origin main
5. From now on: use git push and git pull as usual.

3. SSH Keys — Setup & Management

SSH keys provide a secure and convenient authentication method for Git operations. Steps: generate → load into agent → add to GitHub → test.

Check existing keys (Linux/macOS)

```
ls -al ~/.ssh
```

Check existing keys (PowerShell)

```
Get-ChildItem $env:USERPROFILE\.ssh
```

Generate a new key (recommended: ed25519)

```
ssh-keygen -t ed25519 -C "your.email@example.com"
```

If you need RSA for compatibility

```
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
```

Start the SSH agent & add key (Linux/macOS/Git Bash)

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
```

Start the SSH agent & add key (PowerShell, OpenSSH)

```
Start-Service ssh-agent
ssh-add $env:USERPROFILE\.ssh\id_ed25519
```

Show public key (copy & paste to GitHub)

```
Linux/macOS: cat ~/.ssh/id_ed25519.pub
PowerShell: type $env:USERPROFILE\.ssh\id_ed25519.pub
```

Test connection to GitHub

```
ssh -T git@github.com
# Expected: a greeting with your GitHub username or a success message indicating the connection works
```

Multiple accounts (use ~/.ssh/config)

```
Example ~/.ssh/config:

Host github.com-personal
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519_personal

Host github.com-work
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519_work

Then clone or push using the Host alias: git clone git@github.com-personal:USER/Repo.git
```

Change passphrase for a key

```
ssh-keygen -p -f ~/.ssh/id_ed25519
```

Remove a key from the agent (single)

```
ssh-add -d ~/.ssh/id_ed25519
```

Remove all keys from the agent

```
ssh-add -D
```

Delete key files (be careful)

```
rm ~/.ssh/id_ed25519 ~/.ssh/id_ed25519.pub
```

4. Passphrases — Why & How

A passphrase protects your private SSH key with an additional secret. If someone obtains your private key file, they cannot use it without the passphrase.

Advantages

• Protects against theft of the private key file (on-device or via backup).
• Increases security without extra hardware.

Drawbacks / trade-offs

• You must enter the passphrase, or rely on an ssh-agent that caches it for the session.
• For automated CI/CD jobs, use deploy keys or tokens instead of passphrase-protected user keys.

Change or remove passphrase

```
ssh-keygen -p -f ~/.ssh/id_ed25519
```

Press ENTER to remove the passphrase (not recommended). Prefer using a secure passphrase and an ssh-agent.

5. Project: 'pytasks' — Step by Step

Assumption: You have a GitHub account, Git installed, and at least one SSH key registered with GitHub.

Complete initial setup (compact)

1) Install Git (Windows: Git for Windows; Linux: sudo apt install git / sudo pacman -S git; macOS: brew install git or Xcode CLI tools).
2) Configure user: git config --global user.name "Your Name" and git config --global user.email "you@example.com".
3) Ensure SSH key exists and is added to GitHub: cat ~/.ssh/id_ed25519.pub → add to GitHub.

Practical commands (compact example)

```
mkdir pytasks
cd pytasks
git init
echo "# PyTasks" > README.md
git add README.md
git commit -m "Initial commit"
```

Mini Project: Python & Git — workflow example

1. Create project: mkdir git-demo && cd git-demo
2. Initialize Git: git init
3. Create a file: echo 'print("Hello Git!")' > hello.py
4. Stage: git add hello.py
5. Commit: git commit -m 'Add hello.py'
6. Edit the file: echo 'print("Version 2")' >> hello.py
7. Check status: git status
8. Commit changes: git add . && git commit -m 'Update hello.py'
9. View history: git log --oneline
10. Create & switch to branch: git checkout -b feature
11. Add new file & commit: git add new.py && git commit -m 'Add new feature'
12. Switch back to main & merge: git checkout main && git merge feature

6. Typical Git workflow while you work

Create a feature branch

```
git switch -c feature/add-cli
```

Edit files

```
Use your editor (e.g. code . or nano app.py)
```

View changes

```
git status
```

Stage changes

```
git add app.py
```

Commit

```
git commit -m "feat: add CLI for tasks"
```

Switch to main

```
git switch main
```

Merge feature

```
git merge feature/add-cli
```

Add remote (if needed)

```
git remote add origin git@github.com:YOUR_USER/pytasks.git
```

Rename default branch to main (if needed)

```
git branch -M main
```

Push to GitHub

```
git push -u origin main
```

Get updates from remote

```
git pull
```

Temporarily stash work

```
git stash  # later: git stash pop
```

7. Troubleshooting & Quick Reference

Common issues

Authentication error on push: Ensure your SSH key is added to GitHub and the ssh-agent is running. Test with: ssh -T git@github.com
Merge conflict: git status shows conflict files → edit them manually → git add → git commit
Error 'Permission denied (publickey)': Check ssh-add -l and compare your public key (cat ~/.ssh/id_ed25519.pub) with the key listed on GitHub.

Useful quick commands

```
Show compact log: git log --oneline --graph --decorate --all
List stash: git stash list
Apply specific stash: git stash apply stash@{0}
Remove a local branch: git branch -d feature-branch
Force remove a branch: git branch -D feature-branch
```

End — Good luck experimenting. If you want an A3 poster layout, color variants, or a translated printable handout, tell me which format you prefer.