

Purple Squirrel In a Nutshell

A Desktop Quick Reference

Part I: Developer Tools

Terminal & Shell Commands

Navigation

Command	Description
pwd	Print working directory
ls -la	List all files with details
cd path	Change directory
cd -	Go to previous directory
cd ~	Go to home directory

File Operations

Command	Description
cp src dest	Copy file
cp -r src dest	Copy directory recursively
mv src dest	Move/rename
rm file	Remove file
rm -rf dir	Remove directory (careful!)
mkdir -p path	Create nested directories
touch file	Create empty file

Text Processing

Command	Description
cat file	Display file contents
head -n 20 file	First 20 lines
tail -n 20 file	Last 20 lines
tail -f file	Follow file updates
grep pattern file	Search for pattern

grep -r pattern dir	Recursive search
wc -l file	Count lines

Pipes & Redirection

```
# Pipe output to another command
cat file.txt | grep "error" | wc -l

# Redirect output to file
command > output.txt      # Overwrite
command >> output.txt    # Append
command 2>&1              # Stderr to stdout
command &> output.txt     # Both to file
```

Process Management

Command	Description
ps aux	List all processes
top / htop	Interactive process viewer
kill PID	Terminate process
kill -9 PID	Force kill
command &	Run in background
jobs	List background jobs
fg %1	Bring job to foreground

Git Workflows

Configuration

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global init.defaultBranch main
```

Daily Commands

Command	Description
git status	Show working tree status
git add .	Stage all changes
git add -p	Interactive staging
git commit -m "msg"	Commit with message

git push	Push to remote
git pull	Fetch and merge
git fetch	Fetch without merge

Branching

```
git branch          # List branches
git branch feature      # Create branch
git checkout feature      # Switch to branch
git checkout -b feature      # Create and switch
git merge feature      # Merge into current
git branch -d feature      # Delete branch
```

Undoing Changes

```
git checkout -- file      # Discard working changes
git reset HEAD file      # Unstage file
git reset --soft HEAD~1      # Undo last commit (keep changes)
git reset --hard HEAD~1      # Undo last commit (discard)
git revert <commit>      # Create inverse commit
```

Stashing

```
git stash          # Stash changes
git stash list      # List stashes
git stash pop       # Apply and remove stash
git stash apply      # Apply but keep stash
git stash drop       # Remove stash
```

Rebasing

```
git rebase main      # Rebase onto main
git rebase -i HEAD~3      # Interactive rebase last 3
git rebase --abort      # Cancel rebase
git rebase --continue      # Continue after conflict
```

Useful Aliases

```
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.st status
git config --global alias.lg "log --oneline --graph --all"
```

Package Managers

npm (Node.js)

Command	Description
npm init -y	Initialize project
npm install pkg	Install dependency
npm install -D pkg	Install dev dependency
npm install -g pkg	Install globally
npm uninstall pkg	Remove package
npm update	Update all packages
npm run script	Run package script
npm list	List installed packages
npx command	Run without installing

pip (Python)

Command	Description
pip install pkg	Install package
pip install -r requirements.txt	Install from file
pip uninstall pkg	Remove package
pip freeze > requirements.txt	Export dependencies
pip list	List installed
pip show pkg	Package info

Cargo (Rust)

Command	Description
cargo new project	Create new project
cargo build	Build project
cargo build --release	Build optimized
cargo run	Build and run
cargo test	Run tests
cargo add pkg	Add dependency
cargo update	Update dependencies

Homebrew (macOS)

Command	Description
brew install pkg	Install package
brew uninstall pkg	Remove package
brew upgrade	Upgrade all
brew list	List installed
brew search term	Search packages
brew doctor	Check system
brew cleanup	Remove old versions

IDE Shortcuts (VS Code)

General

Shortcut (Mac)	Shortcut (Win)	Action
Cmd+Shift+P	Ctrl+Shift+P	Command Palette
Cmd+P	Ctrl+P	Quick Open file
Cmd+,	Ctrl+,	Settings
Cmd+B	Ctrl+B	Toggle sidebar
Cmd+J	Ctrl+J	Toggle terminal

Editing

Shortcut (Mac)	Shortcut (Win)	Action
Cmd+D	Ctrl+D	Select next occurrence
Cmd+Shift+L	Ctrl+Shift+L	Select all occurrences
Option+Up/Down	Alt+Up/Down	Move line up/down
Shift+Option+Up/Down	Shift+Alt+Up/Down	Copy line up/down
Cmd+/	Ctrl+/	Toggle comment
Cmd+Shift+K	Ctrl+Shift+K	Delete line
Cmd+Enter	Ctrl+Enter	Insert line below

Navigation

Shortcut (Mac)	Shortcut (Win)	Action

Cmd+G	Ctrl+G	Go to line
F12	F12	Go to definition
Shift+F12	Shift+F12	Find all references
Cmd+Shift+0	Ctrl+Shift+0	Go to symbol
Cmd+\	Ctrl+\	Split editor

Multi-cursor

Shortcut (Mac)	Shortcut (Win)	Action
Option+Click	Alt+Click	Add cursor
Cmd+Option+Up/Down	Ctrl+Alt+Up/Down	Add cursor above/below
Cmd+Shift+L	Ctrl+Shift+L	Cursor at all occurrences

Part II: API Reference

REST Conventions

HTTP Methods

Method	Purpose	Idempotent	Body
GET	Read resource	Yes	No
POST	Create resource	No	Yes
PUT	Replace resource	Yes	Yes
PATCH	Partial update	Yes	Yes
DELETE	Remove resource	Yes	No

Status Codes

Code	Meaning
200	OK - Success
201	Created - Resource created
204	No Content - Success, no body
400	Bad Request - Invalid input
401	Unauthorized - Auth required
403	Forbidden - No permission
404	Not Found - Resource missing

409	Conflict - State conflict
422	Unprocessable - Validation error
429	Too Many Requests - Rate limited
500	Server Error - Bug or failure
503	Service Unavailable - Overloaded

URL Patterns

```

GET   /users          # List users
POST  /users          # Create user
GET   /users/:id       # Get user
PUT   /users/:id       # Replace user
PATCH /users/:id       # Update user
DELETE /users/:id      # Delete user

GET   /users/:id/posts # User's posts (nested)

```

Request Headers

```

Authorization: Bearer <token>
Content-Type: application/json
Accept: application/json
X-Request-ID: <uuid>

```

GraphQL Patterns

Query Structure

```

query GetUser($id: ID!) {
  user(id: $id) {
    id
    name
    email
    posts(first: 10) {
      edges {
        node {
          id
          title
        }
      }
    }
  }
}

```

Mutations

```
mutation CreateUser($input: CreateUserInput!) {
  createUser(input: $input) {
    user {
      id
      name
    }
    errors {
      field
      message
    }
  }
}
```

Fragments

```
fragment UserFields on User {
  id
  name
  email
  createdAt
}

query {
  me { ...UserFields }
  user(id: "123") { ...UserFields }
}
```

Variables

```
{
  "id": "123",
  "input": {
    "name": "John",
    "email": "john@example.com"
  }
}
```

Authentication Patterns

JWT (JSON Web Tokens)

Structure: header.payload.signature

```
// Header
{ "alg": "HS256", "typ": "JWT" }

// Payload
{
```

```
"sub": "user_123",
"iat": 1609459200,
"exp": 1609545600
}
```

Usage:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...
```

OAuth 2.0 Flow

1. Redirect user to provider
→ /authorize?client_id=X&redirect_uri=Y&scope=Z
2. User authorizes, redirected back with code
← /callback?code=ABC123
3. Exchange code for token
POST /token { code, client_id, client_secret }
→ { access_token, refresh_token }
4. Use access token
Authorization: Bearer <access_token>

API Keys

```
# Header
X-API-Key: sk_live_abc123

# Query parameter (less secure)
GET /api/data?api_key=sk_live_abc123
```

Rate Limiting

Common Headers

```
X-RateLimit-Limit: 1000      # Max requests
X-RateLimit-Remaining: 999    # Requests left
X-RateLimit-Reset: 1609459200 # Reset timestamp
Retry-After: 60              # Seconds to wait
```

Strategies

Strategy	Description
Fixed Window	N requests per time window
Sliding Window	Rolling window of time
Token Bucket	Tokens regenerate over time

Leaky Bucket	Smooth output rate
--------------	--------------------

Client Handling

```
import time

def api_call_with_retry(url, max_retries=3):
    for attempt in range(max_retries):
        response = requests.get(url)

        if response.status_code == 429:
            wait = int(response.headers.get('Retry-After', 60))
            time.sleep(wait)
            continue

    return response

raise Exception("Rate limited after retries")
```

Part III: Language Cheatsheets

Python Essentials

Data Types

```
# Primitives
x = 42          # int
y = 3.14         # float
s = "hello"      # str
b = True         # bool
n = None         # NoneType

# Collections
lst = [1, 2, 3]           # list (mutable)
tpl = (1, 2, 3)            # tuple (immutable)
st = {1, 2, 3}             # set
dct = {"a": 1, "b": 2}     # dict
```

List Comprehensions

```
# Basic
squares = [x**2 for x in range(10)]

# With condition
evens = [x for x in range(10) if x % 2 == 0]
```

```
# Dict comprehension
squared = {x: x**2 for x in range(5)}
```

Functions

```
# Basic
def greet(name: str) -> str:
    return f"Hello, {name}"

# Default arguments
def greet(name: str = "World") -> str:
    return f"Hello, {name}"

# *args and **kwargs
def func(*args, **kwargs):
    print(args)    # tuple
    print(kwargs)  # dict

# Lambda
double = lambda x: x * 2
```

Classes

```
from dataclasses import dataclass

@dataclass
class User:
    name: str
    email: str
    active: bool = True

    def greet(self) -> str:
        return f"Hello, {self.name}"
```

Error Handling

```
try:
    result = risky_operation()
except ValueError as e:
    print(f"Value error: {e}")
except Exception as e:
    print(f"Unexpected error: {e}")
else:
    print("Success!")
finally:
    cleanup()
```

Context Managers

```

# File handling
with open("file.txt", "r") as f:
    content = f.read()

# Custom context manager
from contextlib import contextmanager

@contextmanager
def timer():
    start = time.time()
    yield
    print(f"Elapsed: {time.time() - start:.2f}s")

```

Async/Await

```

import asyncio

async def fetch_data(url: str) -> dict:
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.json()

async def main():
    results = await asyncio.gather(
        fetch_data("url1"),
        fetch_data("url2"),
    )

```

TypeScript Essentials

Types

```

// Primitives
const num: number = 42;
const str: string = "hello";
const bool: boolean = true;
const arr: number[] = [1, 2, 3];

// Object types
interface User {
    id: string;
    name: string;
    email?: string; // optional
    readonly createdAt: Date;
}

// Union types
type Status = "pending" | "active" | "closed";

```

```
// Generics
function first<T>(arr: T[]): T | undefined {
  return arr[0];
}
```

Type Guards

```
function isString(value: unknown): value is string {
  return typeof value === "string";
}

// Usage
if (isString(value)) {
  console.log(value.toUpperCase());
}
```

Utility Types

```
Partial<T>      // All properties optional
Required<T>     // All properties required
Readonly<T>      // All properties readonly
Pick<T, K>       // Only specified keys
Omit<T, K>       // Exclude specified keys
Record<K, V>     // Object with key type K, value type V
```

Functions

```
// Typed function
function add(a: number, b: number): number {
  return a + b;
}

// Arrow function
const multiply = (a: number, b: number): number => a * b;

// Optional parameters
function greet(name: string, greeting?: string): string {
  return `${greeting ?? "Hello"}, ${name}`;
}
```

Classes

```
class User {
  constructor(
    public readonly id: string,
    public name: string,
    private email: string
  ) {}
```

```
    getEmail(): string {
      return this.email;
    }
}
```

Async/Await

```
async function fetchUser(id: string): Promise<User> {
  const response = await fetch(`/api/users/${id}`);
  if (!response.ok) {
    throw new Error("Failed to fetch");
  }
  return response.json();
}
```

Rust Fundamentals

Variables & Types

```
// Immutable by default
let x = 5;
let mut y = 10; // mutable

// Type annotations
let num: i32 = 42;
let float: f64 = 3.14;
let s: String = String::from("hello");
let slice: &str = "hello";
```

Ownership

```
// Move semantics
let s1 = String::from("hello");
let s2 = s1; // s1 is moved, no longer valid

// Clone for deep copy
let s1 = String::from("hello");
let s2 = s1.clone();

// Borrowing
fn print_len(s: &String) {
  println!("{} {}", s, s.len());
} // s is borrowed, not moved
```

Structs & Enums

```

struct User {
    name: String,
    age: u32,
}

impl User {
    fn new(name: String, age: u32) -> Self {
        Self { name, age }
    }

    fn greet(&self) -> String {
        format!("Hello, {}", self.name)
    }
}

enum Status {
    Active,
    Inactive,
    Pending { reason: String },
}

```

Pattern Matching

```

match value {
    0 => println!("zero"),
    1..=10 => println!("1-10"),
    _ => println!("other"),
}

// if let for single pattern
if let Some(x) = optional {
    println!("{}", x);
}

```

Error Handling

```

// Result type
fn divide(a: f64, b: f64) -> Result<f64, String> {
    if b == 0.0 {
        Err("Division by zero".into())
    } else {
        Ok(a / b)
    }
}

// ? operator for propagation
fn calc() -> Result<f64, String> {
    let x = divide(10.0, 2.0)?;
}

```

```
    Ok(x * 2.0)
}
```

Iterators

```
let nums = vec![1, 2, 3, 4, 5];

let doubled: Vec<i32> = nums
    .iter()
    .map(|x| x * 2)
    .filter(|x| *x > 4)
    .collect();
```

SQL Quick Reference

Basic Queries

```
-- Select
SELECT column1, column2 FROM table WHERE condition;
SELECT * FROM users WHERE active = true ORDER BY name;
SELECT DISTINCT category FROM products;

-- Insert
INSERT INTO users (name, email) VALUES ('John', 'john@example.com');

-- Update
UPDATE users SET active = false WHERE last_login < '2024-01-01';

-- Delete
DELETE FROM users WHERE id = 123;
```

Joins

```
-- Inner join
SELECT u.name, o.total
FROM users u
INNER JOIN orders o ON u.id = o.user_id;

-- Left join (all from left, matched from right)
SELECT u.name, o.total
FROM users u
LEFT JOIN orders o ON u.id = o.user_id;

-- Multiple joins
SELECT u.name, o.total, p.name
FROM users u
JOIN orders o ON u.id = o.user_id
JOIN products p ON o.product_id = p.id;
```

Aggregations

```
SELECT
    category,
    COUNT(*) as count,
    SUM(price) as total,
    AVG(price) as average,
    MAX(price) as max_price
FROM products
GROUP BY category
HAVING COUNT(*) > 5
ORDER BY total DESC;
```

Subqueries

```
-- In WHERE
SELECT * FROM users
WHERE id IN (SELECT user_id FROM orders WHERE total > 100);

-- In FROM (derived table)
SELECT avg_order
FROM (
    SELECT user_id, AVG(total) as avg_order
    FROM orders GROUP BY user_id
) as user_averages
WHERE avg_order > 50;
```

Indexes

```
-- Create index
CREATE INDEX idx_users_email ON users(email);
CREATE UNIQUE INDEX idx_users_username ON users(username);

-- Composite index
CREATE INDEX idx_orders_user_date ON orders(user_id, created_at);

-- Drop index
DROP INDEX idx_users_email;
```

Transactions

```
BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
```

```
COMMIT;  
-- or ROLLBACK; to cancel
```

Purple Squirrel Media Engineering Excellence